トランジスタ技術 SPECIAL

特集 Z80ソフト&ハードのすべて

基礎からマクロ命令を使いこなすまでのノウハウを集大成

No.6

CQ出版社

トランジスタ技術 SPECIAL No.6

CONTENTS

FEATURES

基礎からマクロ命令を使いこなすまでのノウハウを集大成

Z80ソフト＆ハードのすべて

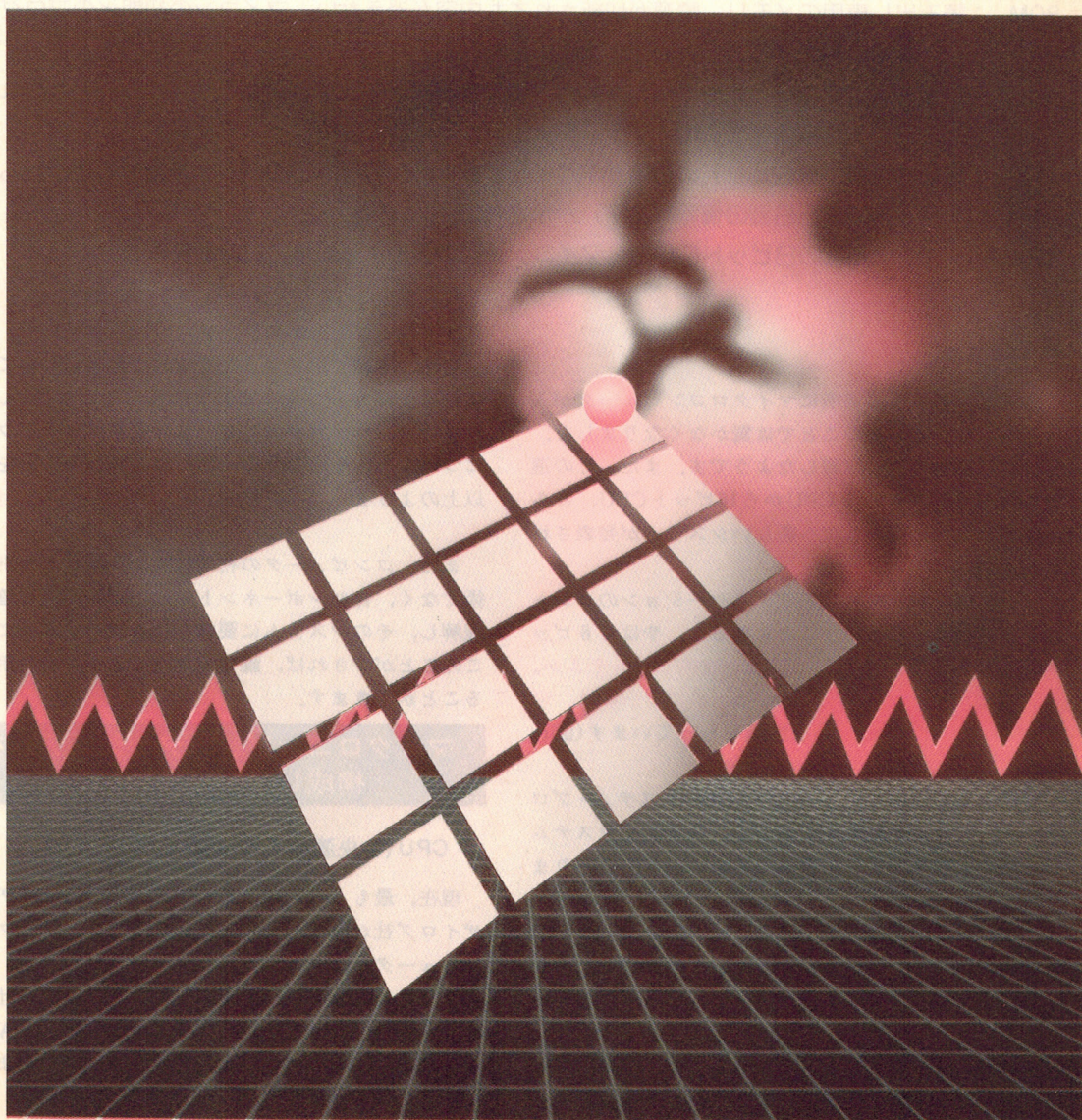
《コラム》

本文イラスト：神崎真理子　表紙デザイン：(株)シーピーユー　簑原圭介

FEATURES

トランジスタ技術SPECIALの第6号は，現在一番使われているZ80とその周辺LSIを取り上げます．システムを設計するうえでは，割り込み技術が重要です．また，作ったソフトを蓄積し，有効に活用するためのマクロ命令，モジュール化の手法も詳しく解説します．

基礎からマクロ命令を使いこなすまでのノウハウを集大成

Z80ソフト＆ハードのすべて

マイコン・システムの基本構成

NEXT

ここでは，マイコン・システムの構成例を示し，どのように命令やデータが流れて処理が行われているかを，わかりやすく説明します．

keywords

クロック：タイミングの基準となるパルス．システム・クロックとしてコンピュータの動作の基準となるタイミング・パルス示す．各CPUによって仕様が定められている．

RAM：一般に読み書きが可能なメモリを示す．高集積度化が進んでいる．

ROM：読み出し専用のメモリ．電源が切断されても内容が消えない．マイコンの初期化のプログラム，専用装置のプログラムの保存などに使われる．

インターフェース：異なった機能をもった装置，回路，素子などの仲立ちをするからくり．それぞれの仕様の差をこのインターフェースが吸収する．

バイト：最小の情報の単位ビットの8倍が1バイト．多くの情報は，このバイト単位で処理される．1バイトで256種の情報（文字）が示される．

8080A：インテル社の8ビットCPU．マイコン革命の元祖，その後Z80が受け継いだ．

● はじめに

最近は，あらゆる場所にマイクロコンピュータが入り込み，もう誰もそのことでは驚かなくなりました．それだけ社会の中に定着したようです．また，その進歩も著しく，8ビットCPUから16ビットCPU，さらに32ビットへと次から次へ新しいシステムが発表されています．

しかし我々が，手軽に各アプリケーションのコントロールに利用するということになると，やはり8ビットCPUでしょう．よほど大規模で複雑なシステムか，特別に高速な処理が要求されているのでない限り，現在の8ビットCPUで十分な性能をもっています（図1-1）．

豊富な周辺用のLSI，各種の開発用のシステム・プログラム，これらを利用することで，基本的なシステムなら誰でも自作できると言っても言い過ぎではありません．

ハードウェア製作の後，そのシステムに，いわば生命を吹き込むというべきソフトウェアの開発も，8ビットCPUを使用したパーソナル・コンピュータで容易に行えます．

以前は，そのようなシステムなしで，ハンド・アセンブルという優雅な手法で，2Kバイトくらいのモニタまで作ったことがあります．

要は，本誌で示すような基本的な事項について，できれば配線も一つ一つ自分で確認しながら，はんだ付けを行い，ハードウェアと一体な基本的ソフトウェアについても，その命令の一つ一つの動作の確認を行う，以上のような実践を通してのみ真の技術が身に付きます．

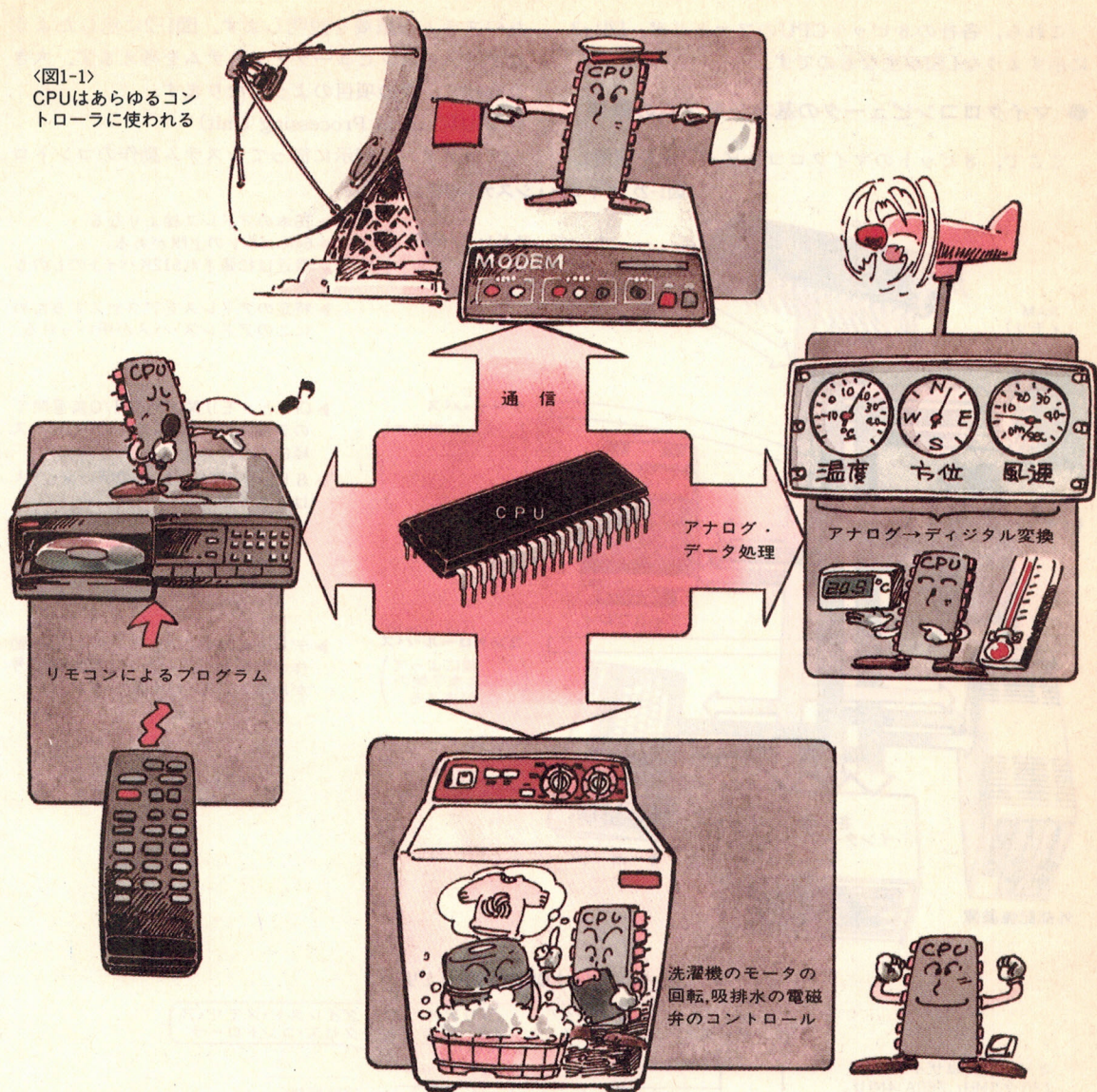
また，コンピュータの利用技術は，決して特別な技術でなく，各コンポーネントの入出力の関係を確実に理解し，そのシステムに要求される機能を明確にする，このことができれば，誰でもコンピュータを利用し作ることができます．

マイクロコンピュータ・システムを構成している基本要素

● CPU（中央演算処理装置）

現在，最もよく利用されている8ビットのCPUは，ザイログ社が開発したZ80と呼ばれているマイクロコンピュータでしょう．

マイクロコンピュータそのものは，インテル社の4ビットCPU 4004に始まり，インテル社の8080Aをきっかけに現在のマイクロコンピュータ革命が始まりました．

〈図1-1〉
CPUはあらゆるコントローラに使われる

その後ザイログ社が，8080Aの命令をすべて含み，そのうえ多くの強化された命令をもったZ80を発表しました．このZ80は，ハードウェア上も多くの優れた特長をもっています．まず，その頃から急激に高集積度化されたダイナミック・メモリのコントロール回路を内蔵しています．そのため，メモリ容量の大きなシステムを容易に作ることができるようになりました．その他にも，命令の実行速度が速い，割り込みの機能が強化されているなどの理由で，瞬く間にZ80が8ビットCPUの標準となってしまいました．

● 基本的な周辺用LSIの概要

マイクロコンピュータ・システムは，CPUだけでなく図1-2に示すようにいくつかの入出力用のインターフェースをコントロールするLSI，ROM，RAMが必要です．その他に，アプリケーションによって，システムのコントロールのためのタイマ，割り込みのコントローラなどが必要です．

これらの周辺用のLSIも，各社から様々な機能をもったものが発売されています．基本的には，各社それぞれのCPUに対するファミリとなっています．

当然のことですが，それらのファミリ内で使用するのが最も効果的です．しかし，ほとんどの場合こうした異なるファミリ間での接続も問題ありません．

とくに，Z80 CPUに対しては，インテル社の8ビット用の周辺用LSIは，まず問題なく接続することができます．

インテル社の8085Aは，豊富な周辺用のファミリをもっていますので，あえてほかのファミリの周辺用のLSIを使用することもないようです．

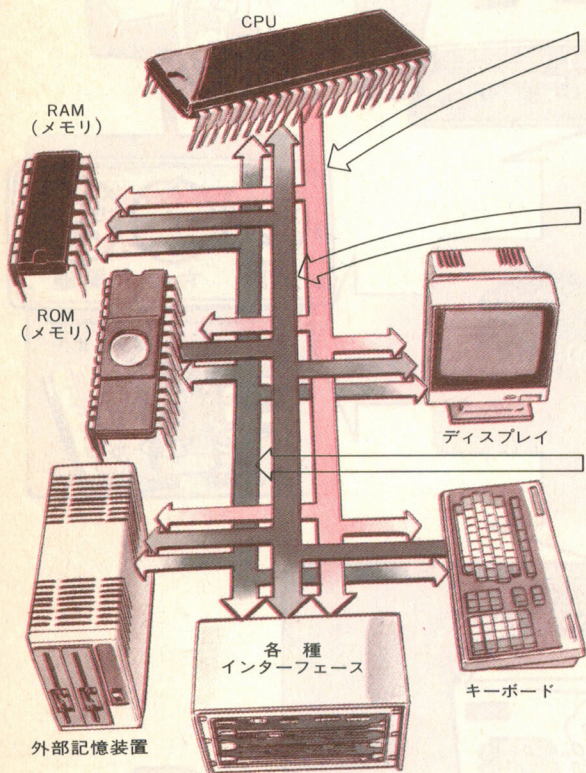
これら，各社の８ビットCPUのファミリは，図1-3に示すようなLSIが主なものです．

● マイクロコンピュータの基本システムの概要

ここで，８ビットのマイクロコンピュータを使用したシステムの概要を説明します．図1-2に示したようなマイクロコンピュータ・システムを考えます．大きく分けて，次の項目のようになります．

▶CPU（Central Processing Unit）

プログラムの指示に従ってシステム動作のコントロ

〈図1-2〉 マイコン・システムの構成

アドレス・バス：
- ▶16本のアドレス線よりなる
- ▶64Kバイトの上限がある
- ▶最近は拡張され512Kバイトのものもある
- ▶特定のアドレスをアクセスするためにこのアドレス・バスが用いられる

データ・バス：
- ▶CPUとメモリ間，CPUとI/O装置間でのデータの伝達は，このデータ・バス経由で行われる
- ▶８ビットCPUでは，このデータ・バスは８本である

コントロール・バス：
- ▶データの移動のタイミング，CPUの動作の制御を行うための各種制御信号が用意されている（図1-9参照）

〈図1-3〉 ８ビットCPUの各ファミリ

オリジナルはザイログ社．Z80：2.5MHz，Z80A：4MHz，Z80B：6MHz，Z80H：8MHzのクロックが使える．ファミリもクロックに応じた速度のものを用いるが，CPUにくらべ低速のものを用いるときには，CPUにウエイトを入れる（後述）

オリジナルはインテル社．マイコン革命のスタートとなった8080Aのファミリで，多くのペリフェラル（周辺装置）をもっている．AM8085A，MSM80C85AのようにC-MOSタイプのものもある．

オリジナルはモトローラ社．同系統のファミリとして，6802，6803，F6808など多くのバリエーションがある．すぐれたアーキテクチャであると評価は高い．

用語解説

- ▶パラレルLSI：８ビットなどのデータを同時に出力または入力することのできる並列入出力用のLSI．
- ▶タイマLSI　：システム・クロックなどを利用して，一定間隔のパルスを作ったり，そのパルスを利用して計数する機能をもったLSI．
- ▶カウンタLSI：外部からのパルスを計数する機能をもつ．そのほかに，入力したパルスを所定の値で分周する機能もあり，タイマと同一の素子に組み込まれている．
- ▶シリアルLSI：1本の信号線で，データを直列に送受信するためのインターフェース．調歩同期式と呼ばれる方法が一般的に使用されている．
- ▶DMA　　　：CPUを介することなく，メモリとメモリ，もしくは外部の素子との間で，直接データのやりとりを行うためのLSI．高速なデータの入出力が行える．

ールを行う中枢部分です．この部分にどのようなマイクロコンピュータをもってくるかでシステムの性格が決まります．

▶メモリ

プログラムおよびデータを記憶しておくために，必ずコンピュータ・システムではメモリが必要です．このメモリには，ROM，RAMの二通りがあります(図1-4)．

(1) ROM

ROM(ロム)は，リード・オンリ・メモリ(Read Only Memory)と呼ばれて，システムをコントロールするためのプログラムを入れておきます．ROMは，電源を切った場合でもメモリの内容が保存されています．したがって，システムの電源の投入と同時にROM上のプログラムが起動し，システムに要求された動作を開始します．

専用システムの場合このスイッチONと同時にシステムがスタートする使いやすさは，不可欠なものです．

〈図1-4〉 メモリの接続

しかし，電源を切っても内容が消えないかわりに，システムに実装されている場合，メモリの内容を変えることはできません．したがって，変数またはシステムが外部から読み取ったデータなどは保存できません．

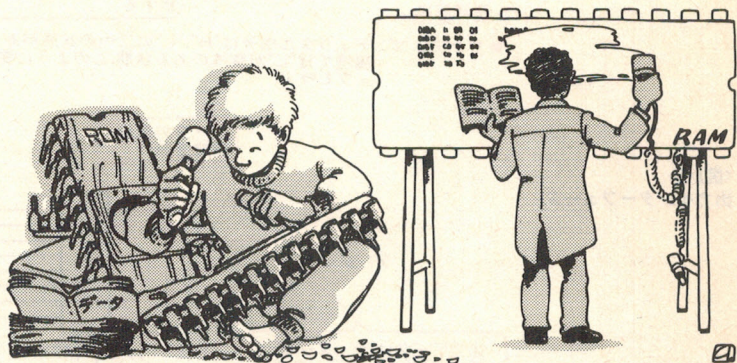
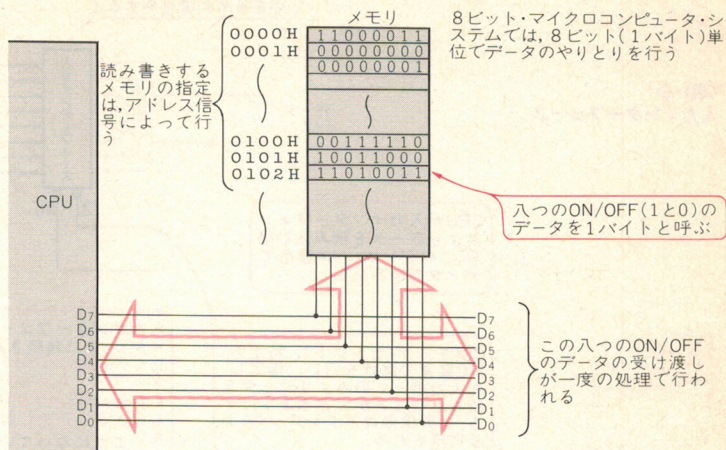
(2) RAM

RAM(ラム)は，ランダム・アクセス・メモリ(Random Access Memory)と呼ばれ，コンピュータ・システムの中で，データまたは各種の変数の一時記憶場所として活用されています．しかし唯一の欠点は，電源を切ると内容が蒸発してしまうことです．この蒸発してしまうことをとらえて，電源のしゃ断で内容の消えるメモリのことを揮発性のメモリと呼んでいます．ROMのことは不揮発性のメモリといいます．

最近は，システムの電源が切断されても，メモリに内蔵されたリチウム電池でデータの保存に必要な電源を供給する，不揮発性のRAMも発売されています．

▶外部とのデータの交換を行う入出力装置の概要

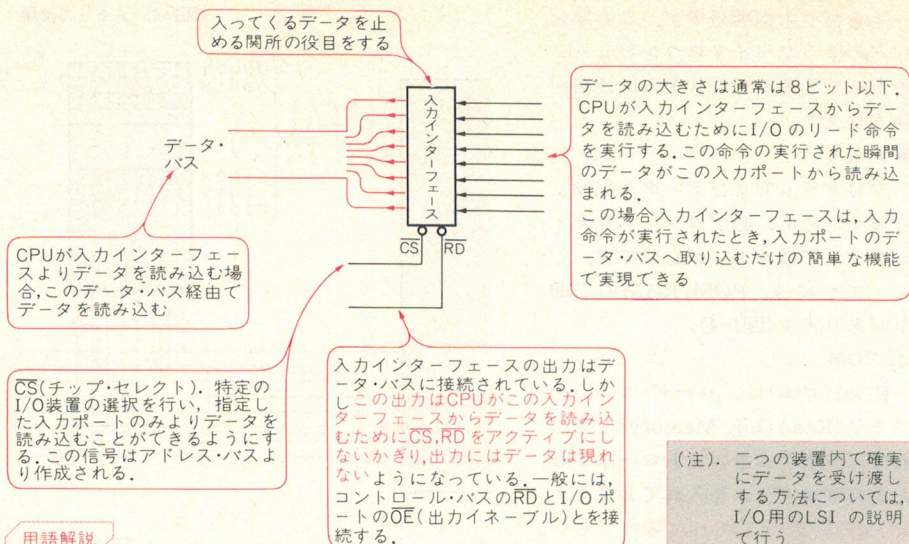
コンピュータのシステムがプログラムの指示に従って行った処理の結果を，なんらかの方法でコンピュータ・システムから外部に示す必要があります．また，処理に必要なデータを外部から取り込む必要もあります(図1-5，図1-6参照)．

このような，機能をもったものとして，パーソナル・コンピュータのディスプレイおよびキーボードなどがあります．

シングル・ボードのコンピュータでは，入出力装置と接続するためのインターフェースのみ用意している場合が多くあります．このような処理のために，ターミナルと呼ばれる専用のディスプレイとキーボードの組み合わせの装置があり，よく利用されています．しかし，このターミナルは案外高価ですので，むしろ若干のプログラムを作り，パーソナル・コンピュータで代用させるとよいでしょう．

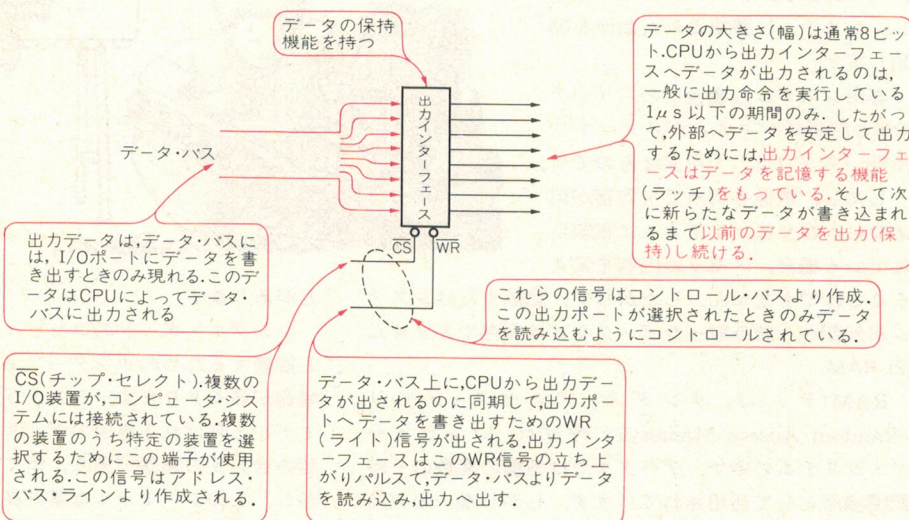
▶システムの動作の基準となるクロック発振回路

コンピュータ・システムの中では，メモリから読み込んだプログラムの各命令を解読し，そのプログラムの指示に従って多様な処理を整然と行わなければなりません．この整然と処理を進行させるための基準になるのがクロックです．このクロックは，それぞれのコンピュータ・システムに応じた特別な仕様をもっています．

しかし，ディジタル・コンピュータと名の付く物なら電卓でもクロックをもっています．電卓をFMラジオのそばに持っていくと，雑音がはいります．これは電卓の内部のLSIをコントロールしているクロックの悪戯です．

〈図1-5〉
入力インターフェース

用語解説

▶アクティブ:ディジタル信号は"H","L"の二つの状態がある.正論理で表現する場合は"H"に意味があり,負論理では"L"が意味のある状態.このように要求に応じた意味のある信号の状態を能動アクティブと呼ぶ

〈図1-6〉
出力インターフェース

用語解説

▶同期する:送信側,受信側で,データの受け渡しを確実にするために,同一の基準となる信号をもとに,有効なデータの範囲を相互で合わせること.

コンピュータで実行される最小の命令も,さらに細かくみればいくつかの基本的な動作に分解されます.

これらの基本的な動作の処理は,CPUに加えられたシステム・クロックによって進められます.したがって,このクロックのスピードがシステムの処理速度を決める大きな要素となっています.Z80も最初に発表された物は,2.5MHzを最大クロックとしていました.しかし最近では,6MHzも普通で,8MHzの物も発表されています.

▶バスはデータ,コントロール情報のハイウェイ

今までに説明した各ブロックは,単独では存在することができません.データやコントロールのための信号を伝える配線が必要です.これは,普通バス方式と呼ばれる配線方法で,各ブロックが結ばれます.これは,バス通りを正確な時刻表に従って運行されるバスによって,データが運ばれることを思い浮かべてください(図1-7).

バスが正確だという点がひっかかるかもしれませんが,コンピュータではこの正確な時刻表がキーになります.そして,このバスは三系統あります.

これだけは

ハードウェアは購入できてもアプリケーション・ソフトは購入できない

知っておきたい

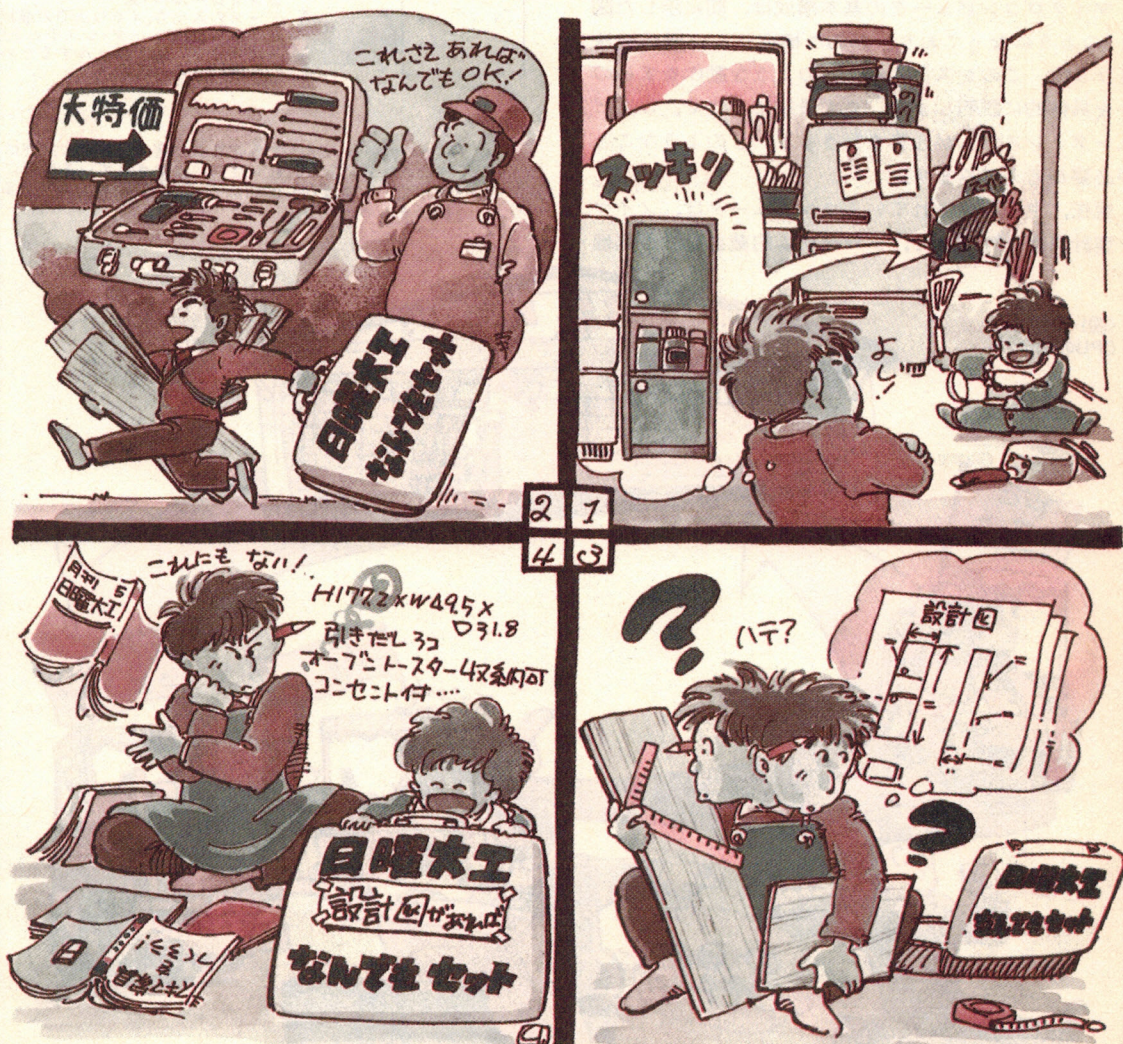
コンピュータを使用するには，必要とするハードウェアを用意するだけでなく，そのハードウェアを制御するためのソフトウェアが不可欠です(図1-A).

ハードウェアとしては，最近ではパーソナル・コンピュータや完成されたボードが多数市販されています.

また，各種の機能をもったオプションでインターフェース用のハードウェアも入手できます．したがって，普通の場合はハードウェアは市販のものを購入することで間に合わすことができます．しかし，そのハードウェアを制御し，必要とする機能を実現するためのソフトウェアは，個々に作成しなければなりません.

とくに最近では，８ビットのマイクロコンピュータは各種のコントローラとして，いろいろな装置，設備に組み込まれることが多くなっています．現在，パーソナル・コンピュータの主流は16ビットCPUを用いたものに変わってしまっています．けれども，８ビットCPUの応用範囲はあらゆる場所に広がり，その機能を十二分に発揮しています.

現在，このマイクロコンピュータの技術者に要求されているのはハードウェアの設計能力だけでなく，要求された仕様を実現するためにソフトウェアのことをも熟知したバランスのとれた能力です.

〈図1-A〉 ハードウェアだけではシステムは動かない

(1) データ・バス

8ビットのデータがCPUとメモリ,I/Oの各素子との間を,このデータ・バスを介して行き来します.

(2) アドレス・バス

16ビットのアドレス情報がCPUからメモリ,I/Oの各素子に出力されます.

(3) コントロール・バス

データの読み書き,メモリ,I/Oのアクセスの区別,その他CPUの状態の制御および表示のための各種のコントロール信号が,それぞれ入出力として設定されています.

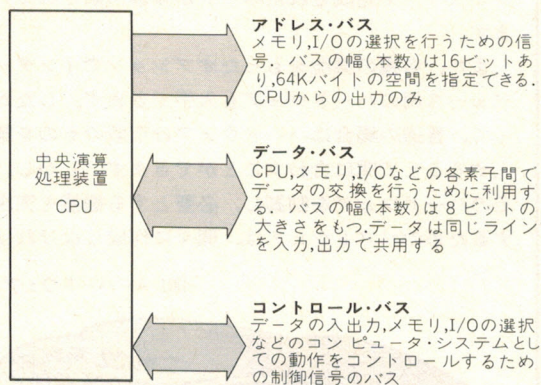
これらについては,のちほど具体的な回路をもとに説明します.

● CPUの命令の具体的な実行シーケンス

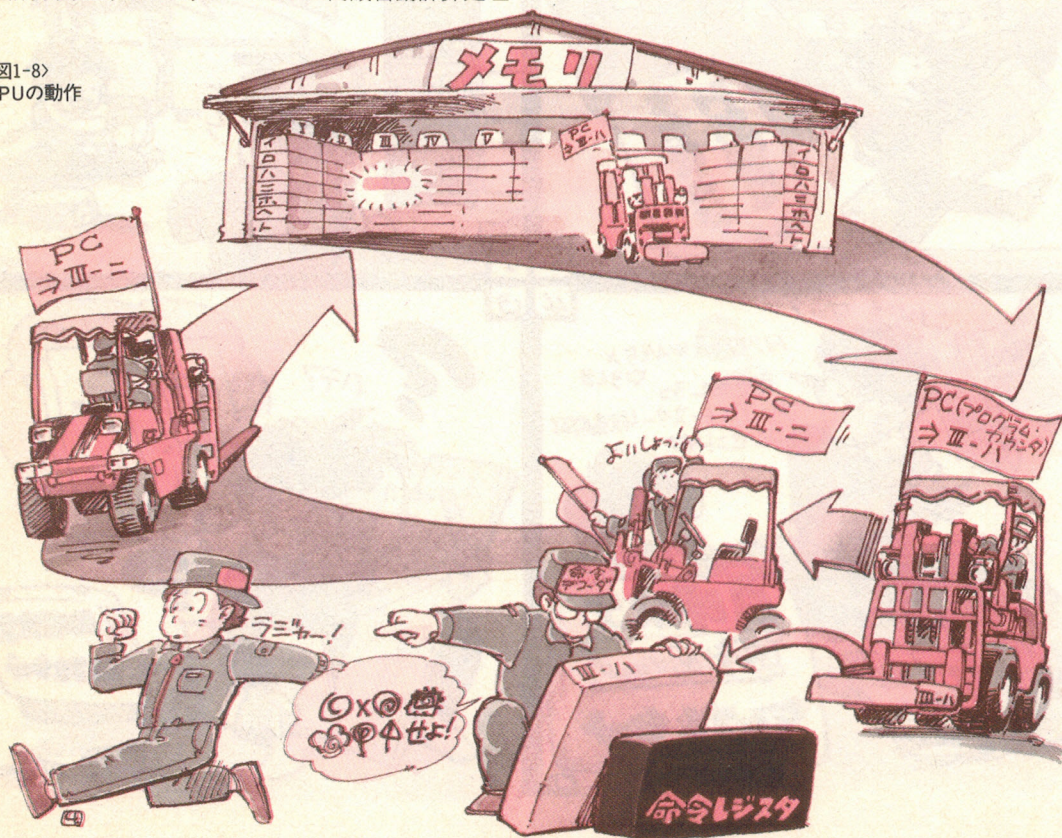
マイクロコンピュータの基本構成は,前に示した図1-2のようになっており,静的な構造はこの構成で決まります.このシステムが,どのような動作をするのかを具体的に説明します.まず最初にここで,コンピュータ・システムが仕事を行うとき,どのようなことが必要か考えてみます.

現在,主に利用されているコンピュータは,ノイマン型計算機と呼ばれて,プログラム内蔵自動計算処理装置とも定義できるシステムです.書き換え可能なプログラムによって制御されることで,このシステムは驚くべき汎用性をもつことができました.また,その処理の高速性はプログラムを内蔵することで得られました.

では,この処理手順はというと次のようになります.

〈図1-7〉CPUからの各種バス

用語解説

バス:複数の信号を入出力する装置が共通な信号線に並列に接続されている.これら並列に接続されている装置に対して,全体または個別にデータの受け渡しが行える機能をもっている.

〈図1-8〉
CPUの動作

(1) プログラムの格納場所より実行すべき命令を読み取る
(2) 読み取った命令を解読する
(3) 命令に従って処理を行う．データの移動，演算，外部とのデータのやりとりなどがある
(4) 命令の処理が終わったら，次の命令を読む準備をする．プログラム・メモリを順次に読むのが原則．しかし，プログラムの実行順序を変更する制御命令がある場合は，その命令に従って次に実行すべき命令のあるアドレスの値にプログラム・カウンタ(PC)の中の値が変更される
(5) 次の命令を読み取り以後同様な動作を繰り返す

このような動作を，正確にタイミングを刻むシステム・クロックと同期を取りながら実行していきます．

その実行のようすを図1-8で説明します．まず，CPUはプログラムの最初の命令を読み込みます．この命令を読み込むサイクルを，オペコード・フェッチ・サイクル(OP code Fetch Cycle)と呼びます．その命令を解読して，忠実に実行します．

この実行サイクルをエクスキュート・サイクル(Execute Cycle)と呼びます．その命令の実行が終了したなら次の命令を読み込み，解読し実行し，それを繰り返します．

これだけは知っておきたい

K, Mバイト

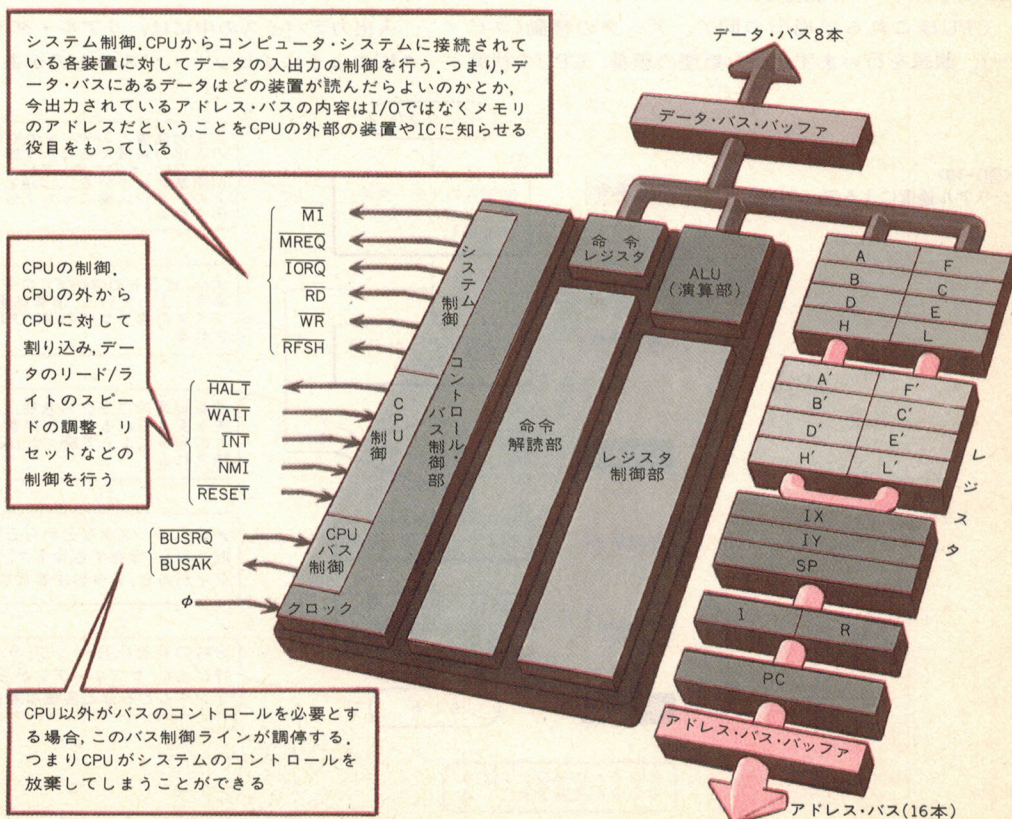
コンピュータのメモリの大きさを表すとき何Kバイトという表現を用います．しかし，このKは普通の単位系では1000を表しますが，ここでは2の累乗となる1024のことを示します．

表1-Aに，それらの関係を示します．

〈表1-A〉
実際のバイト数

1バイト	1バイト
1Kバイト	1,024バイト
4Kバイト	4,096バイト
16Kバイト	16,384バイト
64Kバイト	65,536バイト
128Kバイト	131,072バイト
256Kバイト	262,144バイト
512Kバイト	524,288バイト
1Mバイト	1,048,576バイト

これらの処理を行うCPUの論理的な内部構造を図1-9に示します．プログラムを作成するには，この中のレジスタに関する部分については十分理解しておかなければなりません．

〈図1-9〉
Z80CPUの
論理構造

命令のある場所のアドレスを示すのがプログラム・カウンタ(Program Counter＝PC)と呼ばれるレジスタです．命令のフェッチ・サイクルでは，このPCの値がまずアドレス・バス・バッファにセットされ，メモリ中の命令のあるアドレスから命令コードを読み取ります．

読み込まれた命令コードは，命令レジスタに運ばれたのち，命令解読部で解読作業が行われます．この解読作業は，オペコード・フェッチ・サイクルの最後T_4ステートと呼ばれるときに行われます．解読された命令に従って，レジスタ，コントロール・バスの制御を行います．

このように，命令の読み取り(フェッチ)，解読，実行を繰り返してプログラムが処理されます．

電源の投入時や，CPUがリセットされたとき，このPCの値が0となっています．したがって，一番最初に実行されるプログラムは0000H番地からにセットしておきます．

● マイコン・システムでは各種バスを通してデータが受け渡しされる

次に，CPUはどのような処理ができるのか概観します．図1-2に示したように，CPUの処理の対象となるデータは，メモリ，入出力装置，レジスタと呼ばれるCPU内のデータの記憶領域となります．

CPUはこれらの場所の間で，データの移動(コピー)，演算を行います．また処理の順番，CPUの状態制御の命令も用意されています．

しかし，ここで用意されている命令は，データ処理のための最も基本的なものだけです．何か具体的な仕事をしようとすると，その仕事を実行するための最も細部の手順にまで分解し，その一つ一つにCPUのもっている命令を割り当てて，プログラミングをしなければなりません．

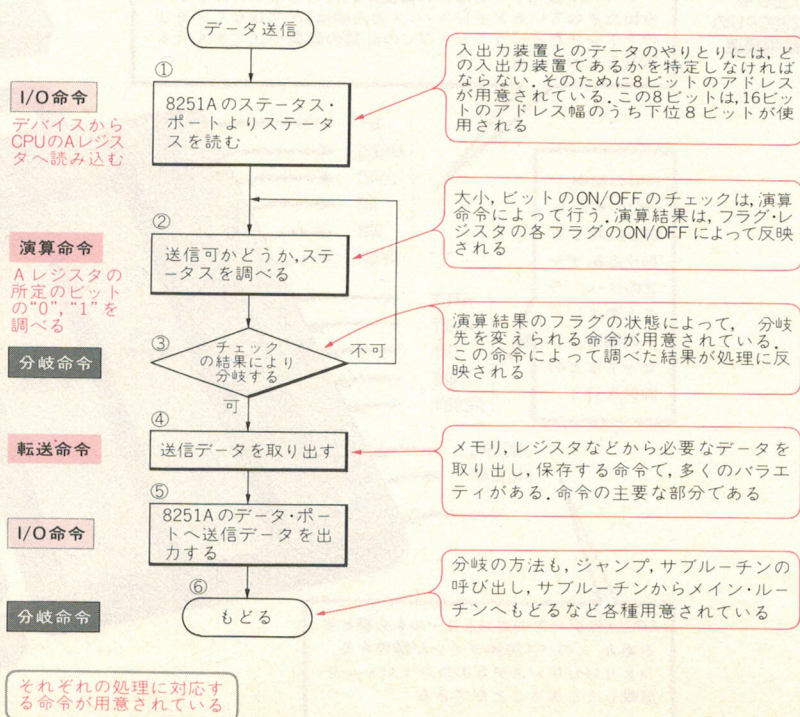
そのため，基本的な命令つまりアセンブラでプログラムすると，思いのほかプログラムのステップが長くなります．

このように常に細部の手順までプログラムしなければならないのが，アセンブラの欠点でありまた長所となっています．細部までプログラマが指示できるので，どんなことでも可能となるのです．

入出力デバイスとのデータの交換の例

CPUには，メモリ以外に外部の装置またはオペレータとのコミュニケーションを行うために，入出力デバイスが接続されます．この入出力デバイスの多くは，8ビットCPUのデータ・バスの大きさに合わせて，ビット単位で入出力が行える出入口をもっています．この出入口をポート(port)と呼び，入力ポート，データ・ポートなどとそのポートの性格に応じた名で呼んでいます．

入出力デバイスの中には，リアル・タイム・クロックICのように4ビット・バスのものがあったり，A-

〈図1-10〉
シリアル通信によるデータ送信

D/D-Aコンバータのように，12〜16ビット・バスのものもあります．これらの扱い方は，8ビットの場合と同じです．つまり，上位4ビットを隠して読み書きするとか，2度にわたって読み書きします．

この入出力デバイスの具体的なデータのやりとりの例として，パソコン通信に使用されるシリアル通信用デバイスの8251Aに対する処理を考えてみます．

データのやりとりを行う場合，常に次のことは考慮しなければなりません．

(1) 相手がデータを受け取れるかどうか
(2) 相手からデータが渡されたのかどうか

この二つの項目をチェックするため，多くの入出力デバイスは書き込みの可否，受信データの有無などを示す手段をもっています．受信データの有無，その素子の状態を示すポートを，データを入出力するためのポート以外にもっています．

8251Aに対してデータを出力する場合も，8251Aの状態をチェックするステータスを読み取り，送信可能であるかどうかを調べます．送信可能であれば，送信データをデータ・ポートに書き込みます．前のデータが送信中であったり，相手側の都合で送信できない場合は，送信できるようになるまで待ちます．

プログラムはメモリに格納されています．そしてプログラム・カウンタと呼ばれるCPU内部のレジスタで，次に実行する命令の格納されているアドレスが指示されています．CPUは，このプログラム・カウンタの指示するアドレスにあるデータを，プログラムの命令として読み込み解釈します．そして，その命令に従った処理を行います．

ここでは，図1-10に示す仕事の流れ図(フローチャート)に従って，次々に実行されるものとして説明します．この命令が読み込まれるようすを図1-11に示します．

この命令読み込みのタイミングを，オペコード・フェッチ・サイクル(M1サイクル)と呼びます．プログラムの実行速度が問題にならない場合は，このようなタイミングについては考慮しなくてもすみます．しかし，後に出てくる割り込み処理などでは，処理スピードを詳細に検討しなければならない場合が起こってきます．

これらCPUが，バス上で命令，データを処理する方法は，非同期バス方式と呼ばれる方法で，処理対象のスピードに合わせた処理が行われます．

メモリなどは高速で処理できます．しかし，一般にI/Oデバイスでは，メモリに比べて読み書きの速度がかなり遅いものが多いので，I/O命令での読み書きのサイクルが長くなっています．

非同期バスでは，このように素子の速度に応じて処理速度を変えられます．一方，バスの処理を同一のクロックに同期して処理するコンピュータのシステムもあります．そのような場合，接続されている最も遅い

〈図1-11〉命令読み込みのタイミング

各信号名
→横は時間軸
各信号ラインのこの色の部分がアクティブである
T_1 T_2 T_3 T_4
"H"
"L"
クロック
A_0〜A_{15}の16本の"H"と"L"を同時に表現しており，データがここで切り替わる
アドレス・バス A_0〜A_{15}
メモリの読み書きでは$\overline{MREQ}$を使用する．I/Oの読み書きでは$\overline{IOREQ}$が使用される
コントロール・バス $\overline{MREQ}$ $\overline{M1}$ $\overline{RD}$
命令コードの読み込み時には$\overline{M1}$がアクティブになる．データの読み書きではインアクティブ
読み込みでは$\overline{RD}$，書き込みでは$\overline{WR}$がアクティブになる
命令コードの読み込みはこの期間で行われ，CPU内部で命令の解読および処理が行われる
データ・バス D_0〜D_7
① アドレス・バスに，次に処理するメモリまたはI/O装置のアドレスを出力し，所定の素子の選択をする
② アドレス・バスの出力に続き，メモリの読み書きか，I/Oの読み書きか，読み取りか，書き込みかなどの制御を行う信号が出力される
③ アドレス・バス，制御バスの指定に従ったデータがバス上に現れ，入出力などの処理が行われる
この範囲はメモリより正しい命令データが現れる．その前の状態は，前の値が残っていたり，不定な値であったりする．
① アドレスが出る
② 制御バスがアクティブになる
③ データの読み書き（命令の読み込み）
この順序で処理が行われる

素子に合わせた処理速度で処理しなければなりません．

Z80は非同期バス・システムですので，バス・サイクルの時間およびその命令の実行速度も合わせて検討します．また，CPUの命令および動作だけでなく，接続されている各ペリフェラル(周辺)素子の使用方法，動作についても理解しなければなりません．本誌でも，そのような観点から基本的なマイクロコンピュータ・システムで必要とする，各ペリフェラルの処理の具体例についても説明を加えます．

図1-10に示す処理は，具体的には次のようになります．

① 最初に読み込んでいる命令は，入出力装置として定義されている通信用のペリフェラル8251Aのステータス・ポートからステータス・データを読み込んでいます．

② 次に，読み込んだデータのビットを調べ，送信可かどうかをチェックしています．

③ 前の命令で調べた結果に従って，それぞれの所定の命令に分岐します．送信可であれば，次にデータ送信の命令に分岐します．送信可でなければ再度ステータスを読む部分にもどり，送信可になるまで繰り返します．

④ 送信可であれば，送信データを取り出します．

⑤ 所定のレジスタに取り出された送信データを，8251Aのデータ・ポートへ書き出します．

実際のプログラムの中では，データの取り出しなどでもう少し工夫されていますが，基本的にはこのようなプログラムで処理されます．

ここで示したように，プログラムの中では様々なタイプの処理が行われています．

コンピュータ・システムを構成する個々の部品

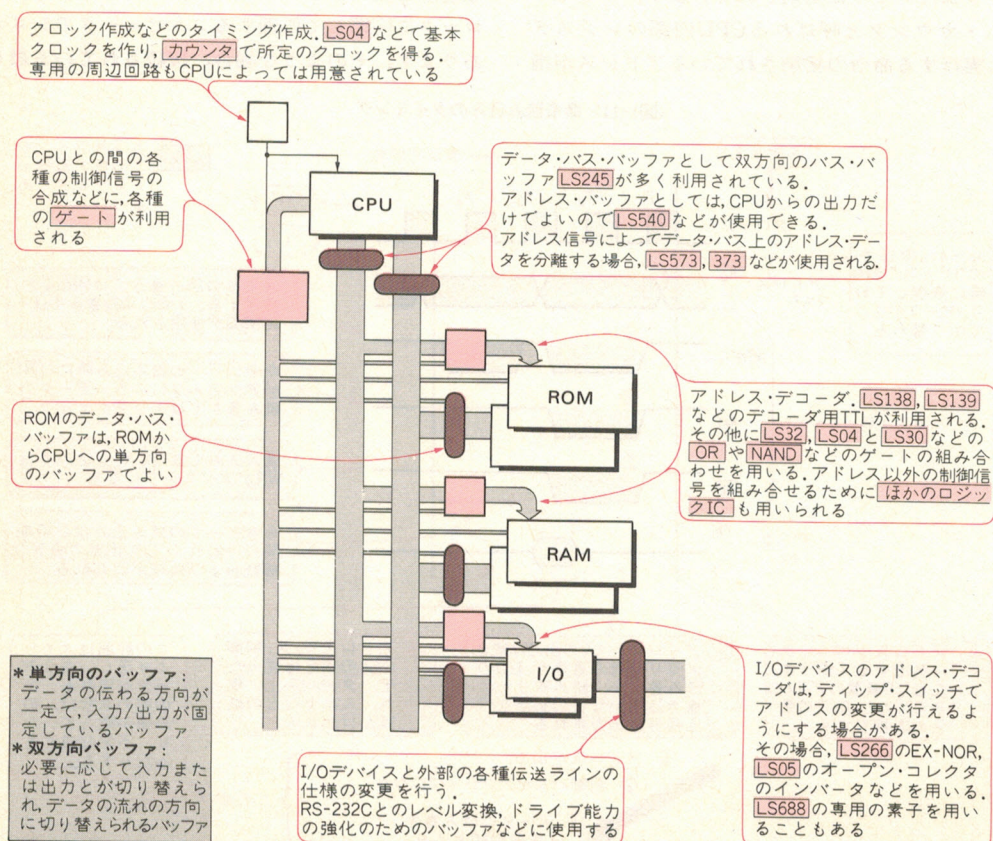
ワンチップCPUといっても，CPU，周辺用のLSIを組み合わせただけでコンピュータ・システムができるわけではありません．

ここでは，それらハードウェアの作成に必要となる各構成部品について説明します．

(1) 各LSIチップ

CPUチップ，インターフェース用の周辺用LSI．これらは高い集積度をもち，それぞれ専用の素子として開発されています．これらについては以後の各章で詳しく説明します．

(2) ロジック用IC

〈図1-12〉 マイクロコンピュータ・システムに利用されるロジックIC

CPU，周辺用LSIの集積度も高くなっています．しかし，それらだけでなく，図1-12に示すように多くの場所で汎用のロジックICが利用されています．

(a) アドレス・デコーダ

ゲートやインバータの組み合わせ，またはアドレス・デコーダ用のロジックICもあります(図1-13，図1-14)．

〈図1-13〉 アドレス・デコーダの基本

アドレスの入力が特定の値のときにのみ，出力がイネーブルになるようにする

アドレス入力

$\overline{A_7}$ 0
A_6 1
A_5 1
A_4 1
$\overline{A_3}$ 0
$\overline{A_2}$ 0
A_1 1
A_0 1

多くのデバイスの選択信号の入力は，"L"でイネーブルとなっている．接続されている素子が選択されたときのみ"L"になるようにする

出力

$\overline{A_0 \cdot A_1 \cdot \overline{A_2} \cdot \overline{A_3} \cdot A_4 \cdot A_5 \cdot A_6 \cdot \overline{A_7}}$

8入力のNAND．入力すべてが"H"のときにのみ出力は"L"になる素子である

選択のときのアドレスの値が"L"のものについては，インバータ(反転出力が得られる)によって選択されたときの値が"H"になるようにする

〈図1-14〉 LS139の内部ロジック

LS139は2入力のアドレス・デコーダに入力バッファと出力イネーブルの制御が付加されたものが2回路入っている．詳細な回路を(b)に示す．

(a) LS139のピン配置

(b) 内部ブロック

デコーダ・マルチプレクサ
機能表

入力			出力
イネーブル入力 G̅	セレクト入力 B	A	$Y_0 Y_1 Y_2 Y_3$
H	X	X	H H H H
L	L	L	L H H H
L	L	H	H L H H
L	H	L	H H L H
L	H	H	H H H L

(c) 真理値表

(b) 制御信号の合成

各デバイス間の制御を行う場合，その制御の論理にしたがってロジックICを組み合わせて制御回路を作成します(図1-15)．

(c) バッファ

CPU，周辺用LSIの多くは，出力のドライブ能力が大きくありません．多くの素子を接続したり，システム外のプリンタなどの機器を動かすときなどの信号の強化に利用されます．

これらロジックICを使用するにあたっての注意点は，大きなものとして次の二つの点です．それぞれ各デバイスのデータ・シートに記述されていますから，設計上の留意点としてください．

(i) AC特性

入力端子に加わった入力信号が出力端子に現れるまでに時間遅れが生じます．多くのデバイスを利用すると無視できない時間となり，CPUと各デバイス間のデータの伝達時間の重要なファクタの一つとなります．LSタイプのNANDゲートで約6 nsあります．バッファやデコーダだと20～40nsになるものもあるので，高速処理(Z80で4 MHz以上のクロックを用いるときなど)では，必ずデータ・シートで確認してください．

(ii) DC特性

ディジタル素子は，最近ではTTL以外にC-MOSまたはVMOSなどと異なった種類のものを混用することが多くなっています．

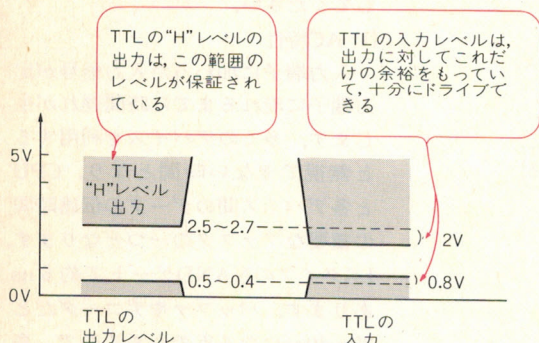
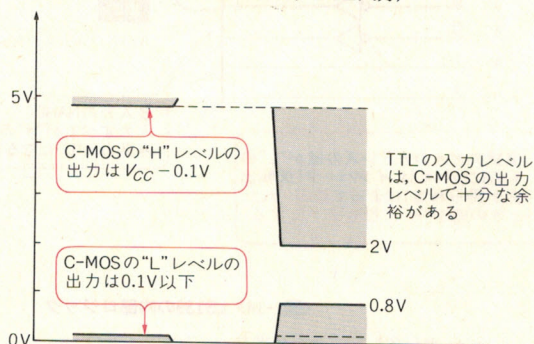
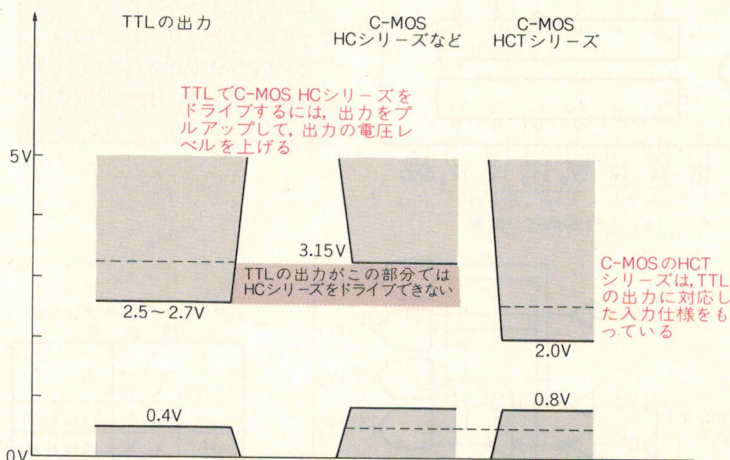
この場合，入出力信号の電圧レベルおよび出力が，出力に共通に接続されている各デバイスをドライブすることができるかどうかのチェックが必要です．図1-16～図1-18を参照してチェックしてください．

(3) トランジスタ，LED

最近では，十分なドライブ能力をもったバッファ用のICが用意されているので，トランジスタを使用する場面はほとんどなくなりました．

しかし50～100mA以上の電源のON/OFFとなると，トランジスタを使用しなければならない場合もあります．また，メモリの電源端子で，

〈図1-15〉
Z80の制御信号から8080Aタイプの制御信号を作る

〈図1-16〉 各素子間の入出力の直流レベル（TTLの出力，TTLの入力）

〈図1-17〉 各素子間の入出力の直流レベル（C-MOS出力，TTL入力）

〈図1-18〉
各素子間の入出力の直流レベル（TTLの出力，C-MOS入力，HCシリーズ）

バッテリとAC電源の切り替えなどに利用されています．

LEDは，信号の状況表示に多く利用されています．

(4) コンデンサ

マイクロコンピュータ・システムで利用されるコンデンサは，ほとんどが電源のバイパス・コンデンサで，時定数用のコンデンサはワンショット・マルチバイブレータに使われるのがほとんどです．

電源のバイパス・コンデンサは，10μF～100μFのタンタル電解コンデンサを基板の電源の入力ラインに，0.1μF～0.001μFの高周波特性の良いセラミック・コンデンサをデバイス数個に対し１個くらいで用意します．ダイナミック・メモリの場合は，電源の供給はこのコンデンサから行うくらいの気持でデバイスと１：

これだけは　知っておきたい

論理回路

ディジタル回路は，“H”，“L”の二つの状態の組み合わせ，演算によって成り立っています．そして論理回路の表現には，MIL(Military standard)記号による記述が一般的です．これら論理演算の基本は論理和(OR)，論理積(AND)，否定(NOT)などです．これらの処理の表現を図1-Bに示します．

この論理回路の表現も，正論理と負論理での二つの表現方法があります．この正論理とは，信号のレベルが“H”のときに意味があることを示します．アドレス・バスに出てくるアドレス信号などは正論理で表現されています．

負論理とは，信号のレベルが“L”のときに意味がある動作を行い，信号レベルが“H”のときには休止状態となります．この休止状態をインアクティブであると表現します．この信号が有意な状態になったとき，この信号はアクティブであると表現します．

信号名の上に―をつけて，その信号が負論理の信号であることを示します．MIL記号ではこの負論理の信号を小さな白い丸で示します．

コンピュータ・システムで用いられる多くの制御信号がこの負論理となっています．

図に示すように同一の素子でも，正論理とみるか負論理と考えるかでANDがORになったりもします．

しかし，MIL記号でそれぞれの信号の正負の論理の区別も含めて記述すれば，各信号間の関係も容易に読み取ることができます．

〈図1-B〉論理素子(ゲート)の表示方法

主なTTL	正論理	負論理	真理値表 (正論理→H=1, L=0／負論理→H=0, L=1) A B Y
7404	NOT $Y=\overline{A}$	NOT $Y=\overline{A}$	L ― H H ― L
7407	バッファ $Y=A$	バッファ $Y=A$	L ― L H ― H
7400	NAND $Y=\overline{AB}$	INVERT-OR $Y=\overline{A}+\overline{B}$	L L H L H H H L H H H L
7408	AND $Y=AB$	INVERT-NOR $Y=\overline{\overline{A}+\overline{B}}$	L L L L H L H L L H H H
7402	NOR $Y=\overline{A+B}$	INVERT-AND $Y=\overline{A}\,\overline{B}$	L L H L H L H L L H H L
7432	OR $Y=A+B$	INVERT-NAND $Y=\overline{\overline{A}\,\overline{B}}$	L L L L H H H L H H H H
74266	エクスクルーシブNOR $Y=AB+\overline{A}\,\overline{B}$	インクルーシブAND $Y=\overline{A}B+A\overline{B}$	L L H L H L H L L H H H
7486	エクスクルーシブOR $Y=\overline{A}B+A\overline{B}$	インクルーシブNAND $Y=AB+\overline{A}\,\overline{B}$	L L L L H H H L H H H L

ゲート(GATE)とは，本来 門，水門などの意味

電子回路では，一つ以上の入力に対して，一つの出力信号をもつ回路を指す

1の割合で接続します．

(5) 抵抗

抵抗類は，オープン・ドレインまたはオープン・コレクタ出力の負荷抵抗，または各信号のバスのプルアップ/プルダウンに利用されます．電力容量は1/4または1/8Wの小型のもので十分です．

バス・ラインのプルアップの実装にあたっては4～8個の抵抗を同一パッケージにおさめた集合抵抗を利用すると実装が簡単です．

基板外と接続されているバス・ラインのプルアップ抵抗は，無信号時の信号レベルの固定以外に，外部からの静電気などのノイズなどによる破壊の防止にもなります．

(6) スイッチ類

コンピュータ・システムで最も目につくのはキーボード・スイッチです．

それ以外に基板上にディップ・スイッチが多く用いられています．アドレスの設定，使用状況のモードの指定などに利用できます．スイッチのON/OFFによって，信号のレベルの設定に対応します．

ディップ・スイッチによっては，BCDコードを出力するものもあります．これを使うとアドレス設定などで，ミスをなくすることができます．

(7) 基板，コネクタ，ソケット

これだけは　知っておきたい

余った端子の処理

素子の端子のうち，使わずに余ったものは，次のように処理するのを原則としてください．

● 入力端子が二つあるのに一方しか使用しない場合

その動作に応じて“H”か“L”に固定します．使用している端子をドライブしている素子に余力があれば，入力を共通にします．

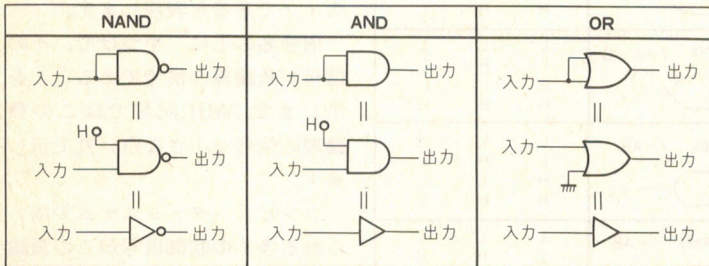
固定する場合，“L” にするにはGNDに接続します．“H” に固定する場合は，74LSシリーズは電源端子と入力端子の耐圧が同じ7Vですから，電源に直接接続します．しかし，スタンダード・タイプまたはマルチ・エミッタ入力の端子の耐圧は5.5Vまでですので，電源投入時などの瞬間に耐圧を越えることのないように，1kΩ程度の抵抗を介して電源に接続します．

〈図1-C〉余った端子の処理例

NAND	AND	OR
入力 → 出力 ＝ H, 入力 → 出力 ＝ 入力 → 出力	入力 → 出力 ＝ H, 入力 → 出力 ＝ 入力 → 出力	入力 → 出力 ＝ 入力 → 出力 ＝ 入力 → 出力

● 使用していない入出力端子の処理

TTLの場合は，オープン(何も接続しない状態)でも特に問題は起きません．

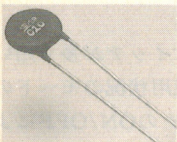
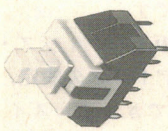
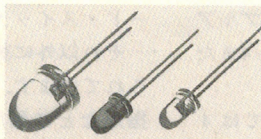
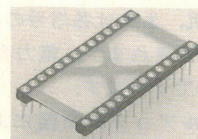
C-MOSの素子の場合は，入力インピーダンスが非常に高いので，オープン状態では論理レベルが一定になりません．また，C-MOSの出力は，入力レベルが中間付近のときに大きな電流が流れます．通常の動作時は，スイッチング動作で中間レベルの時間が短いので問題ありませんが，ノイズで長い時間中間レベルにあると，過電流のために素子が破壊される可能性があります．したがってC-MOSの場合は，使用していないゲートの入力端子は電源かGNDに固定します(直接接続する)．出力端子は，どちらの場合もオープンのままで，ほかに何の影響も与えません．

現在，コンピュータ・システムでは，多層の基板が用いられ実装密度が上がっています．テストなどでは，両面基板に手配線でも実用になります．

ソケットは，ROMなどのように，差し替えの必要な素子には不可欠な部品です．

(8) 電源

コンピュータ・システムの電源は，現在では特別なものでないかぎり小型のスイッチング・レギュレータを利用しています．各素子の省電力化および集積度の向上により，素子数も減少し電源の負担は極めて少なくなっています．また電源の種類も5Vのメインに，＋12Vの2電源または8インチのFDD(フロッピ・ディスク・ドライブ)では24Vが使われるくらいです．

そのほかに，P-ROMの書き込み，RS-232Cのインターフェースなどで，これらと異なった電源電圧を必要とする場合もありますが，ほとんどが5VからDC-DCコンバータで取り出すことがよく行われています．

以上本誌ではこれ以上詳しくふれませんが，CPU，LSIの機能およびソフトウェア以外にも多くの関連技術の集積されたものとしてコンピュータ・システムが実現されています．

▲コンデンサ

▲スイッチ

▲LED

▲ICソケット

フォト・カプラ▶

〈写真1〉部品の外観

アセンブラの基礎

NEXT

まずZ80の基本的な命令体系について説明し，CPUがどのように解読して実行するかということを，わかりやすく説明します．

keywords

レジスタ：CPUチップ内のデータの一時的な記憶域．データ処理の対象となる．
PC　：プログラム・カウンタ．CPUが次に実行する命令のアドレスをセットするレジスタ．命令が読み込まれるたびに更新される．
ニモニック：機械語命令を記憶しやすい有意の文字列に対応させた命令コード．
アドレッシング：命令の処理の対象を具体的に示すためのアドレスの設定．修飾方法．
スタック：本来は棚の意味がある．渡すデータを棚に積み上げ，相手は上から順番に取り出す．サブルーチン処理でデータの受け渡しに利用される．
OPコード　：命令コード．その命令の処理内容を示す．オペランドのない命令もある．
オペランド：命令の処理の対象を示す．命令はOPコードとオペランドからなる．
フラグ：Flag，旗を示す．旗の上げ下ろしで状態を示すように，ビットのON/OFF で指定された状態を示すのに使用される．

本章では，Z80のシステムを実際に動かすために必要な，ソフトウェアに関連した事項について説明します．プログラムを作るときには，ハードウェアの設計上の物理的な問題とは別に，データの流れおよびそのデータの処理の流れの論理的な眺めが必要となります．

具体的には，Z80 CPUの命令の種類，命令の実行の論理的な流れ，メモリ中のデータあるいは命令を特定するためのアドレッシングについて説明します．また，命令の実行速度の推定についても説明します．

Z80の論理構造を示すレジスタの基本的な機能

● アーキテクチャ

Z80の論理的な構造(アーキテクチャ)は，図2-1に示すようになっています．大きく分けると次のようになります．

▶データ，命令が読み書きされる経路となるデータ・バス．データ処理のときに一時的なデータの記憶場所となり，各種の演算のために利用されるレジスタ群．

▶ALU(Arithmetic and Logic Unit：算術論理ユニット)と呼ばれる8ビットの数値演算および論理演算を行う装置．

▶メモリより読み込んだデータを保持する命令レジスタ．命令レジスタの命令を解読し，その命令に従った処理を指令する命令デコーダ，エンコーダ．

▶アドレス・バスにアドレス・バス・バッファの内容が出力される．このアドレスはプログラム中の次に読み込む命令のアドレス，または読み書きの対象となるメモリ中のデータのアドレス，I/O装置のアドレスなど．

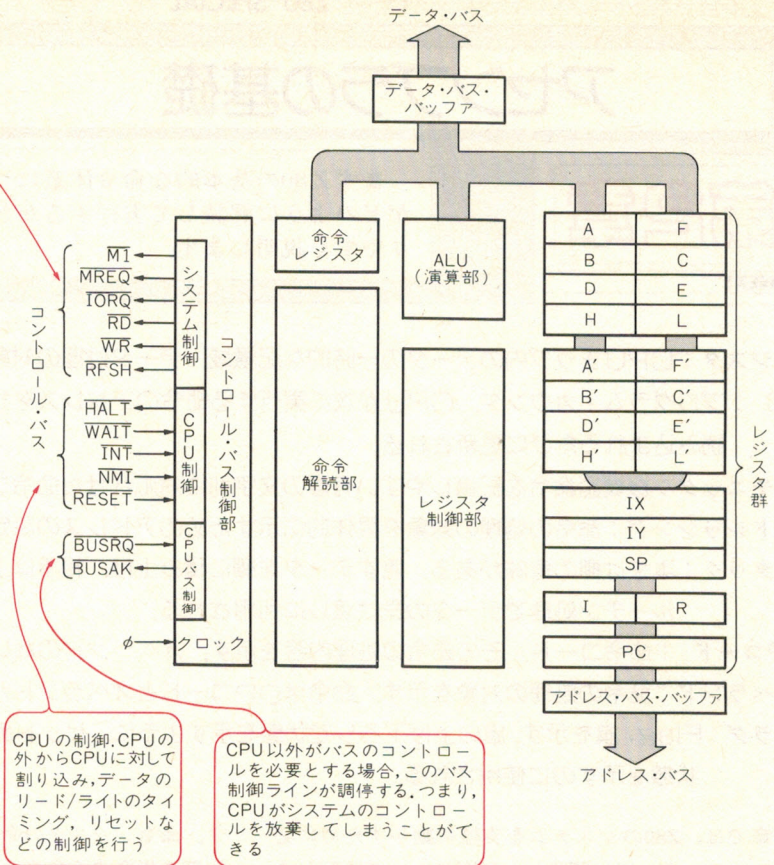
そのほかに，コントロール・バスの制御，読み書きのタイミング，外部の素子との同期のため制御を行う部分があります．

● Z80のレジスタ

プログラマ側から見たZ80の最大の関心事は，データ処理の際に必要となる各レジスタ群のことです．これらレジスタ群について図2-2に示します．

それぞれの働きはAppendix②に詳しく説明しますので，ここでは簡単に説明します．

Aレジスタ(アキュムレータ)は，8ビットの算術および論理演算処理において結果が得られます．B-C，D-E，H-Lとを組み合わせて，16ビットのレジスタとしても利用できる汎用レジスタがあります．16ビットのアドレスのポインタとして主に利用される専用レジスタ，これには，IX，IYのインデックス・レジスタ，スタック処理に利用されるSP(スタック・ポインタ・レ

〈図2-1〉 Z80マイクロコンピュータの論理構造

ジスタ)があります．

PC(プログラム・カウンタ)は，次に実行するプログラムの命令の入っているアドレスを示します．

Iレジスタは，モード2というモードでの割り込みを処理するときに使用します．割り込み処理ルーチンの，先頭アドレスの入っているテーブルのアドレスの，上位8ビットがIレジスタで与えられます．下位8ビットは，割り込み発生時に割り込みを要求したデバイスより与えられます．詳細については，割り込み処理の第8章でさらに説明します．

Rレジスタは，ダイナミック・メモリのリフレッシュ・アドレスを示します．普通のプログラムでは内容を操作することはありません．

そのほかに，汎用レジスタ群がもう1セット用意されています．これらは補助レジスタと呼ばれ，割り込み処理時のレジスタのデータ保存などに利用されます．

アセンブラ

機械語のプログラムの記述には，アセンブラというソフトウェアが使用されています．Z80のアセンブラは体系だった記述法で，命令体系を覚えるのにもそれほど苦労せずにすみます．本章ではザイログ社の表記法によるプログラムの記述方法で説明します．

CP/M80に付属するアセンブラ(ASM.COM)は，インテル社のニモニックで表記されていて，Z80の命令は用意されていません．そのためマクロ定義によってZ80の命令を使用するもの(MAC.COM)，インテル社のニモニックにZ80の命令のニモニックを追加したもの(ZASM.COM)などもあります．

● アセンブラは機械語一つ一つに対応したニモニックをもつ

メモリに格納され，CPUの内部で解釈される機械語の命令は，8ビットのON/OFF(1と0)のかたまりです．

2進数，16進数表示で これらの命令を指定(記述)することもできます．数ステップのプログラムでは，ときには，このような機械語のコードを直接メモリに書き込みデバッグを行うこともあります．しかし，普通のプログラミングでは，この機械語のコードに対応した，アセンブラのニモニック・コードを用いて，プロ

〈図2-2〉
レジスタの種類とその役割

主レジスタ群

A	F
B	C
D	E
H	L

汎用レジスタ

○それぞれペアで，16ビットのレジスタとして使用することもできる．
○アドレスを指定する時にも使用される．その時，上位アドレスはB，D，Hレジスタになる．

A；演算処理の中心になる
F；フラグの状態が保存される
B；256バイトまでのバイト・カウンタとして汎用性がある
HL；16ビット・データ処理の中心になる
ペア・レジスタは，16ビット・データ・レジスタ，アドレス・ポインタとして利用できる．
各レジスタは，8ビットの汎用データ処理に利用される．

インテルの8080A，8085Aと互換性のあるレジスタ群

補助レジスタ群

A′	F′
B′	C′
D′	E′
H′	L′

割り込みサブルーチンなどでレジスタのデータの退避に利用する

専用レジスタ

割り込みベクトル I
メモリ・リフレッシュ R
IX インデックス・レジスタ
IY インデックス・レジスタ
SP スタック・ポインタ
PC プログラム・カウンタ

I；割り込みベクトルの上位アドレスの8ビットを示す．割り込みモードで使用する
詳しくは8章を参照

R；ダイナミック・メモリのリフレッシュ用のアドレス・カウンタとなっている．7ビットで，16K，64Kのダイナミック・メモリのリフレッシュが行える

Fレジスタ

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Fレジスタ	S	Z	／	H	／	P/V	N	C

フラグ・レジスタの各ビットはそれぞれ対応する状態を示すのに使用されている．

フラグの内容

S：サイン・フラグ
　演算の結果が負(D_7＝1)のとき，S＝1
Z：ゼロ・フラグ
　演算の結果がゼロのとき，Z＝1
H*：ハーフ・キャリ
　BCD演算結果の下位4ビットからのキャリ，ボローがあれば，H＝1
P/V：パリティ/オーバフロー・フラグ
　パリティ―奇数ならばP/V＝1，
　偶数ならばP/V＝0
　オーバフロー―オーバフローがあればP/V＝1
N*：加算/減算フラグ
　減算なら　N＝1
C：キャリ/リンク・フラグ
　アキュムレータの最上位ビットからの桁上がりでセットされる
　加算の時 キャリ／減算の時 ボロー があればC＝1
　シフト，ローテイトによって，ビットのチェックを行える
(＊印はプログラマが関与する必要のないフラグ)

グラムは書かれます(図2-3)．

このアセンブラの命令は，そのCPUのもつ命令すべてに対応した表記法をもっています．この表記は**ニモニック・コードと呼ばれる，プログラマから見て，命令の動作の意味がわかりやすい表記法**で定義されています．したがって，アセンブラは，そのCPUがもっている機能を実現することのできるあらゆるプログラムを，わかりやすく書くことができます．

● ニモニックの具体的な例

Z80のニモニックは，次のように三つに分けられます．

(1) 命令部分だけで命令を表現しているもの．
(2) 命令部分と，その命令を修飾する一つのオペランドよりなるもの(オペランドとは，多くの場合，命令の効果のおよぶ対象と考えて間違いない)．
(3) 命令と二つのオペランドとからなるもの．この場

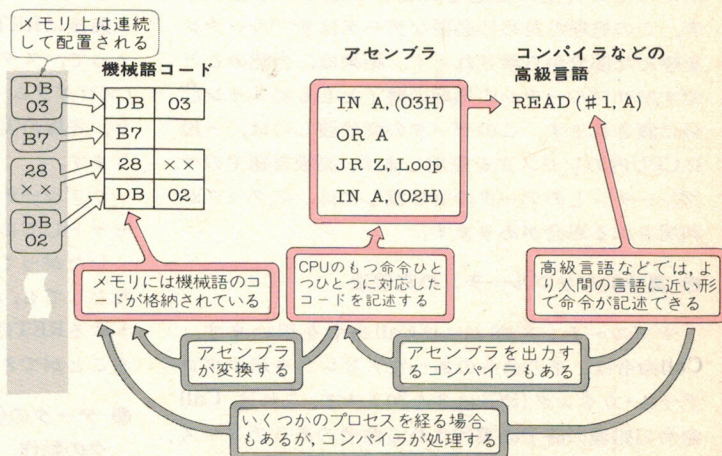
〈図2-3〉 機械語とニモニックの対応

合は，演算などのように二つのデータ間で処理を行う命令．多くの場合，１番目のオペランドが処理の対象となり，結果を求めることのできる対象で，２番目のオペランドは処理を施すデータを示す．

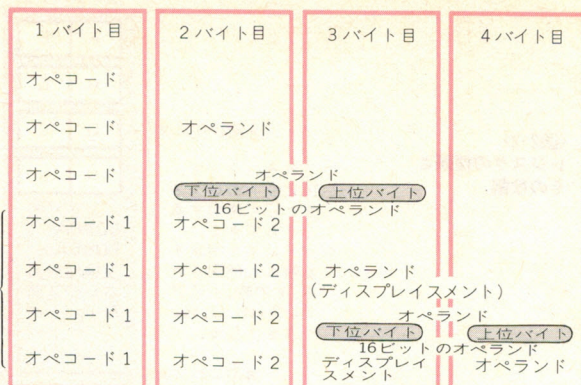
図2-4と図2-5にその具体的な例を示します．

〈図2-4〉 Z80の機械語のコードは１から４バイトとなる

用語解説

▶オペコード：オペレーション・コード．その命令の具体的な操作や動作を示す．
▶オペランド：命令を修飾して，命令の操作の具体的な対象を明確にするところ

● アセンブリ言語の構文

プログラムは，所定の記述法にしたがって記述（コーディング）する必要があります．命令の記述部分は一般に，ラベル，オペコード，オペランド，コメントの順に書かれます（図2-6）．ラベルは，命令コードの置かれている場所を示すために使用されます．ラベルを参照することで，その命令コードが置かれているメモリのアドレスが得られます．アドレスの参照の必要がない場合はラベルも必要ありません．

オペコードとオペランドで，実際の機械語の命令に対応するアセンブラ命令を記入します．命令によって１ないし２個のオペランドをもつ場合と，オペランドをもたない場合があります．

コメントはプログラムの説明を書くためのもので，必ずしも必要ではありません．しかし，このコメントは忘れっぽい自分自身に対するメッセージとしても，必ず付記するよう心掛けてください．

サブルーチン

これだけは知っておきたい

● サブルーチンを利用することでプログラムを機能別分割できる

サブルーチンとは，あるまとまった処理を行う目的で作られたプログラムの部分です．そしてこのサブルーチンは，普通メインのプログラムの複数の場所からその特定の処理を実現するために呼ばれます．この処理のために必要なデータはサブルーチンを呼んだ部分から渡されます．結果は，当然のことですがサブルーチンの処理の終了とともにメイン部分に渡されます．このデータの受け渡しには，一般にCPU内のレジスタを使用します．高級言語でのサブルーチンとのデータの受け渡しには，スタックが利用される場合があります．

● 具体的なサブルーチンの実現法

サブルーチンを呼ぶにはCall命令を用います．Call命令は，その命令のあったアドレスを示すプログラム・カウンタ（PC）に３を加えます．これは，Call命令の処理の終了後実行される命令のあるアドレスとなります．このもどり場所を示すアドレスをスタックに保存してから，オペランドで示されたサブルーチンの入口にジャンプします．Call命令によってここまでは自動的に行われます．

サブルーチン内で使用され，メイン部でのデータが破壊されるレジスタが生じる場合があります．このデータの保存が必要な場合，これもPUSH命令によってスタックに保存します．

サブルーチンでの所定の処理の終了後，POP命令で保存してあったレジスタのデータをスタックより回復（元にもどす）します．

最後にRET命令を実行します．RET命令の実行によって，スタックに保存してあったもどり番地がPC（プログラム・カウンタ）にセットされます．これにより，先程の実行されたCall命令の次の命令の処理に移ります．

サブルーチンを呼ぶほうでもどり番地をスタックにセットしてあります．

したがって，サブルーチン側はどこから呼び出されたとしても，たんにスタック上の戻り番地をPCにセットするRET命令の実行だけで，確実にメイン部にもどることができます．

● データの保存，Call命令などで利用されるスタックの動作

スタックとは，一連のメモリの領域を用いてデータの受け渡しのために，一時的にデータを保存するバッ

〈図2-5〉アセンブリ言語の構文(ザイログ)

☆**表記法**

Z80のアセンブリ言語の表記法つまりニモニックには，ザイログの表記法のほかに8080Aのインテルの表記法を独自に拡張したものがある．これは8085Aなどを使う人にとっては便利だが，ニモニックの数も多く，まちがいもでやすいので，Z80は簡潔なザイログ表記法を学ぶほうがよい．

一般に，高級言語とよばれる，FORTRAN, Pascal, BASICなどには，実際の問題解決(プログラミング)のために必要な抽象化された命令が用意されています．これら高級言語の命令は，各CPUのもつ命令によってプログラムされています．そして，Z80とか6809などと個々のCPUによって，たとえ命令が異なっていたとしても，コンパイラは標準の共通な命令を，CPUの独自の命令の組み合わせに展開します．

したがって，コンパイラを使用すると，プログラマはCPUの違いを意識することなくプログラムすることができます．そのために，マイクロコンピュータから大型の汎用コンピュータまで，同様なプログラムが

ファのことをさします．

このデータ構造は，図2-Aに示すようにデータを積み上げてあるようなものです．データは上にしか積み上げられません．また取り出す場合も，上から順番に取り出すしかありません．

このようなものをスタックと呼び，最後にセットしたデータが最初に取り出されることから，LIFO(Last In First Out)メモリとも呼ばれます．

Z80のシステム上では，具体的にSP(スタック・ポインタ・レジスタ)を用いて，スタック上の最後にセットされたデータのあるメモリ・アドレスが示されます．

スタックへのデータのセットは，PUSH, Call命令，割り込みの受け付け時に行われます．スタックからのデータの取り出しは，POP, RET, RETI命令によって行われます．

このスタックの取り扱いが，プログラミング技術の重要な項目の一つとなっています．

〈図2-A〉スタック・ポインタの働き

```
0000  31FF03            LD   SP, 03FFH   スタック・ポイントを決める①
0003  213402            LD   HL, 0234H   HLレジスタにアドレス0234Hを入れる
0006  3E00     LOOP:    LD   A, 00H      Aレジスタに00Hを入れる
0008  CD0E00            CALL INPUT       次の命令のアドレス(000BH)をスタックへ入れて，コール先の命令へとぶ②
                                         〔スタック・ポインタは03FDH〕
000B  77                LD   (HL),A      Aレジスタの内容を,HLレジスタで示されるアドレスのメモリへ入れる
000C  18F8              JR   LOOP        LOOPへジャンプ
000E  E5       INPUT:   PUSH HL          HLレジスタの内容をスタックへ退避させる③
                                         〔スタック・ポインタは03FBH〕
000F  DBFF              IN   0FFH        ポートFFHの内容をAレジスタへ読み込む
0011  E1                POP  HL          HLレジスタの内容を復帰させる④
                                         〔スタック・ポインタは03FDH〕
0012  C9                RET              スタックから次に実行する命令のアドレスをPCへ戻して，メイン・ルーチンへもどる⑤
                                         〔スタック・ポインタは03FFH〕
```

H	02
L	34

番地	内容
03FA	
03FB	34
03FC	02
03FD	0B
03FE	00
03FF	

PUSH HL ③ (03FB–03FC)　④ POP HL
CALL INPUT ② (03FD–03FE)　⑤ RET
① 03FF

動きます．ただし，その機器に依存する入出力および数値表現できる大きさ(数値表現に利用するメモリの大きさ)などの特別な場合を除きます．

● アセンブラでプログラムを書くのは，そのCPUの特徴をいかすためである

抽象化された命令を実現するために，各マイクロコンピュータは，それぞれ特徴ある命令体系，アーキテクチャ(内部の構成)をもっています．それらの特徴を生かしたプログラムを作るには，アセンブラを使用するしかありません．また，そのマイクロコンピュータを確実に理解するには，アセンブラでプログラムしてみるのが最も簡単な方法です．

一般にマイクロコンピュータ・システムは，そのシステムを適切に運用するためのオペレーティング・システムをもっています．図2-7,図2-8に示すように，ハードウェアに密着した部分には，キーボードからの文字データの入力，コンソール画面へのデータの表示，プリンタ，ディスク・ドライブとの入出力など，いずれの処理でも必要となる機能が用意されています．

これら，基本となる機能をもとに，ファイル処理などよりアプリケーションに近づいた処理の層が形成されます．このへんくらいまでを一般的なDOS(Disk Operating System)と呼ばれるシステムが受け持ちます．

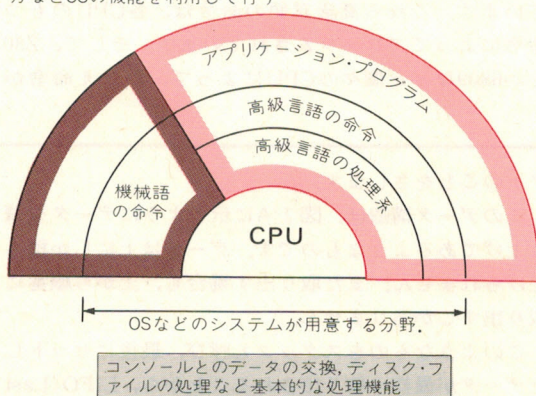
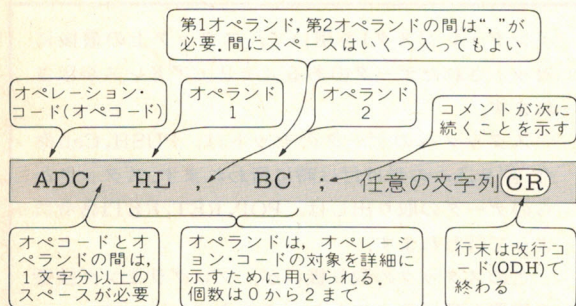
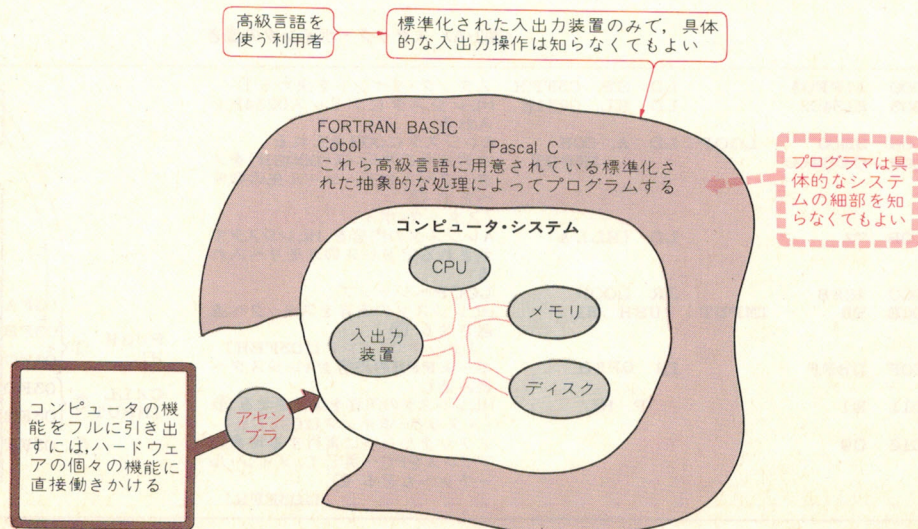
入力された文字列を，数値データとして処理したり，演算処理したりなどとなると，コンパイラなどの高級言語処理の受け持ちとなります．

このように，いくつかに階層化されたそれぞれのシステムを，ユーザは必要に応じて使い分けてアプリケーションを作成します．

そのようにしてアプリケーションを開発するうえで，ハードウェアを直接制御する必要が生じる場合がしばしばあります．そのときは，アセンブラで，DOSなどの核の部分に新たな機能を追加することになります．

なお，ディスクを扱わないようなシングル・ボード・コンピュータなどでは，たんに，モニタなどと呼ばれます．

〈図2-8〉 アプリケーションのプログラム開発

アプリケーション・プログラムの開発は，各システムが用意したディスク・ファイルの処理，コンソールへの表示，キーボードからの入力などOSの機能を利用して行う

〈図2-6〉 アセンブリ言語の構文

〈図2-7〉 コンピュータを取りまくソフトウェアの状況

Z80の命令の基本的な機能概要

Z80の命令は，インテル8080Aの命令体系に新たな命令の追加されたスーパセットとなっています．したがって8080Aの命令はすべて実行することができます．つまり，8080A用に開発されたプログラムが，そのまま何の変更もなく実行することができるわけです．

CP/Mといわれるオペレーティング・システムは，本来インテル8080A用のシステムでした．現在，Z80をCPUとした多くのパーソナル・コンピュータやワープロで使用されています．

最近では，CP/M用のアプリケーション・プログラムの中にZ80独自の命令を使用し効率を上げているものもあります．この場合，8080AのCPUのシステムではこれら効率を上げたプログラムは，動かすことはできません．このように8080AとZ80では一方向への互換性しかありません．

Z80のニモニック・コードは，**表2-1**に示すように数十種類のオペレーション・コード（オペコード）があります．そして，このオペレーション・コードは，その命令の対象をより明確にするために，一つないし二つのオペランドをもっている場合があります（**図2-5参照**）．

このオペランドとなるものも決まっていて，**表2-2**に示すようになっています．

▶レジスタ名

オペランドとして８ビット，16ビット（ペア・レジスタ）

▶フラグの状態

フラグ・レジスタの各フラグの状態を示すオペランドが用意され，条件付きの分岐命令に使用される

▶数値

８ビット，16ビットの数値が使用できる．この数値の表記は２進数，10進数，16進数，定数，変数なども記述できる．

▶ラベル

プログラムの命令コード，またはデータの格納されている場所などを示すためにラベルが用意されている．分岐命令の分岐先，データの転送命令などにも使用される．

命令に先立って記述され，後に'：'が付いている．次に示された命令などのコードが，セットされたメモリのアドレスの値として利用する（詳しくは第２章Appendix①参照）．

Z80の命令を分類するには，いろいろな分け方がありますが，筆者の

〈表2-2〉 オペランド
（オペコードにさらに細かい意味づけをするためのもの，特定の予約語，数値，ラベルなど）

予約語
▶ ８ビットのレジスタ名：A，B，C，D，E，H，L，I，R
▶ 16ビットのレジスタ名（８ビットのペア・レジスタを含む）：IX，IY，SP，AF，BC，DE，HL
▶ 補助レジスタ・ペア名： AF'，BC'，DE'，HL'
▶ フラグ状態名：C（キャリ），NC（ノン・キャリ），Z（ゼロ），NZ（ノン・ゼロ），M（マイナス），P（プラス），PE（パリティ偶数），PO（パリティ奇数）

オペランドの記号	
▶ r	：レジスタA，B，C，D，E，H，Lのいずれかを指す．実際の操作は，そのレジスタの内容について行われる
▶（HL）	：レジスタ・ペアHLの内容で指定されるメモリの内容を示す
▶ n	：１バイトの値で，$0 \leqq n \leqq 256$
▶ nn	：２バイトの値で，$0 \leqq nn \leqq 65535$
▶ d	：１バイトの値で，$-128 \leqq d \leqq 127$．間接アドレッシングにおけるディスプレイスメントを示す
▶（nn）	：２バイトの値nnで指定されるメモリの内容を示す
▶（n）	：１バイトの値nで指定される入出力ポートの内容を示す
▶ b	：０から７までの範囲の値．ビットを表す
▶ e	：１バイトの値で，$-126 \leqq e \leqq 129$．相対アドレッシングにおけるディスプレイスメントを示す
▶ cc	：条件ジャンプJP，JR，CALL，RET命令などにおけるフラグの状態を示す
▶ qq	：レジスタ・ペアBC，DE，HL，AFのいずれかを指す
ss，dd	：レジスタ・ペアBC，DE，HL，SPのいずれかを指す
pp	：レジスタ・ペアBC，DE，IX，SPのいずれかを指す
rr	：レジスタ・ペアBC，DE，IY，SPのいずれかを指す
▶ s	：r，n，（HL），（IX+d），（IY+d）のいずれかを指す
▶ m	：r，（HL），（IX+d），（IY+d）のいずれかを指す

〈図2-9〉 演算命令の要約

〈表2-1〉ニモニックのオペレーション・コード（▭；オペランドのない命令）

●転送命令

ロード命令	
LD	レジスタ，メモリ間のデータ転送（8ビット，16ビット）
PUSH	レジスタの内容をスタックへプッシュする（16ビット）
POP	スタックの内容をレジスタへポップする（16ビット）
交換命令	
EX	レジスタ・ペア同士の内容をそれぞれ交換する
EXX	レジスタ・ペアBC, DE, HLの内容をBC′, DE′, HL′の内容と交換する
ブロック転送命令	
LDI	(DE)←(HL), DE←DE+1, HL←HL+1, BC←BC−1
LDIR	(DE)←(HL), DE←DE+1, HL←HL+1, BC←BC−1, BC=0まで繰り返す
LDD	(DE)←(HL), DE←DE−1, HL←HL−1, BC←BC−1
LDDR	(DE)←(HL), DE←DE−1, HL←HL−1, BC←BC−1, BC=0まで繰り返す
CPI	A−(HL), HL←HL+1, BC←BC−1
CPIR	A−(HL), HL←HL+1, BC←BC−1, A=0またはBC=0まで繰り返す
CPD	A−(HL), HL←HL−1, BC←BC−1
CPDR	A−(HL), HL←HL−1, BC←BC−1, A=0またはBC=0まで繰り返す

●演算命令

加減算	
ADD	キャリを含まない加算(8ビット，16ビット)
ADC	キャリを含む加算　(8ビット，16ビット)
SUB	キャリを含まない減算(8ビット)
SBC	キャリを含む減算　(8ビット，16ビット)
CP	比較（フラグのみ変化する）(8ビット)
INC	+1する（8ビット，16ビット）
DEC	−1する（8ビット，16ビット）
論理演算	
AND	アキュムレータとの論理積をとる（8ビット）
OR	アキュムレータとの論理和をとる（8ビット）
XOR	アキュムレータとの排他的論理和をとる（8ビット）
汎用算術演算	
DAA	アキュムレータの十進補正
CPL	アキュムレータの内容の1の補数をとる
NEG	アキュムレータの内容の2の補数をとる
CCF	キャリ・フラグの1の補数をとる
SCF	キャリ・フラグをセットする

●ＣＰＵ制御命令

NOP	プログラム・カウンタを1進める
HALT	CPUを停止させる
DI	INT割り込みを無効にする(IFF_1, IFF_2をリセットする)
EI	INT割り込みを有効にする(IFF_1, IFF_2をセットする)
IM	割り込みモードを0〜2にセットする

●ローテイト，シフト命令

ローテイト	
RLCA	アキュムレータの内容を左へローテイトする
RLA	アキュムレータの内容を左へローテイトする
RRCA	アキュムレータの内容を右にローテイトする
RRA	アキュムレータの内容を右にローテイトする
RLC	オペランドの内容を左へローテイトする
RL	オペランドの内容を左へローテイトする
RRC	オペランドの内容を右にローテイトする
RR	オペランドの内容を右にローテイトする
シフト	
SLA	オペランドの内容を左へシフトする
SRA	オペランドの内容を右にシフトする
SRL	オペランドの内容を右にシフトする
RLD	A［7 4｜3 0］　［7 4｜3 0］(HL)
RRD	A［7 4｜3 0］　［7 4｜3 0］(HL)

●ビット操作命令

BIT	オペランドで指定されたビットの補数をZフラグに入れる
SET	オペランドで指定されたビットをセットする
RES	オペランドで指定されたビットをリセットする

●分岐命令

ジャンプ命令	
JP	オペランドの内容をPCにロードし，次の命令はそこから始まる．無条件と条件付きがある
JR	オペランドで指定されたディスプレイスメントだけ離れた番地へジャンプする．条件つきもある
DJNZ	Bレジスタの内容を−1し，0でなければJRと同様にジャンプし，0ならば次の命令を実行する
コール，リターン命令	
CALL	コール命令の次の命令の番地をスタックにプッシュした後，オペランドで示された番地へとぶ．条件付きもある
RET / RET(条件付き)	スタックからコール命令の次の命令のアドレスをPCにポップして元のプログラムにもどる．条件付きもある．
RETI	割り込みサービス・ルーチンの終了時に使用するRET命令
RETN	ノンマスカブル割り込みに対するサービス終了時に使用するRET命令

●入出力命令

入力命令	
IN	それぞれオペランドで指定されたポートからレジスタへデータを入力する
INI	(HL)←(C), B←B−1, HL←HL+1
INIR	(HL)←(C), B←B−1, HL←HL+1, B=0まで繰り返す
IND	(HL)←(C), B←B−1, HL←HL−1
INDR	(HL)←(C), B←B−1, HL←HL−1, B=0まで繰り返す
出力命令	
OUT	それぞれオペランドで指定されたポートへレジスタからデータを出力する
OUTI	(C)←(HL), B←B−1, HL←HL+1
OTIR	(C)←(HL), B←B−1, HL←HL+1, B=0まで繰り返す
OUTD	(C)←(HL), B←B−1, HL←HL−1
OTDR	(C)←(HL), B←B−1, HL←HL−1, B=0まで繰り返す

〈図2-10〉 データの転送

独断によってZ80の命令を大別すると次のようになります.

(1) **演算命令**

8ビットの論理演算,算術演算があります.算術演算は,HLレジスタをアキュムレータとして16ビット・データの処理も行えます.算術演算は,加減算のみで積および除算は加減算を用いてプログラムしなければなりません(図2-9).

(2) **データの転送命令**

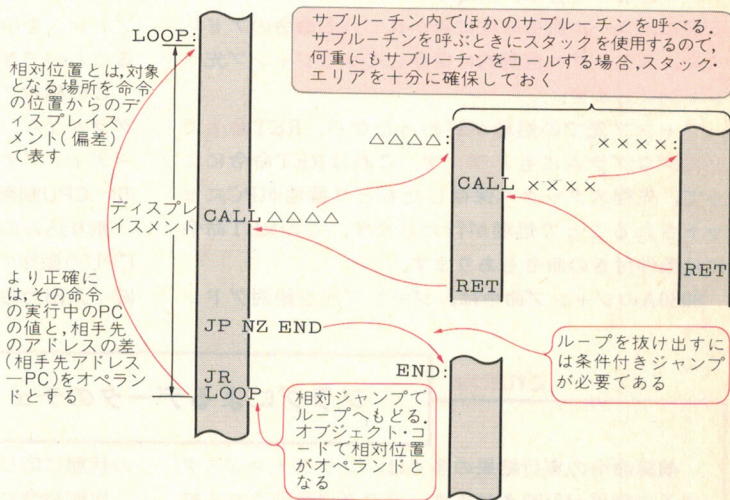
レジスタとレジスタの間でのデータの転送,この場合のデータの転送とはデータの移動ではなく,転送元から転送先へデータがコピーされ転送元のデータは変化しません(図2-10).

レジスタとメモリ間の転送,この場合のメモリのデータの示し方も多くの方法があります.具体的にはアドレッシングの項で説明します.また,レジスタやメモリへ直接 数値を代入することもできます.

(3) **ローテイト/シフト/ビット操作命令**

8ビットのレジスタおよび,HLレジスタまたはIX,IYレジスタによって指定されたメモリの内容のローテイト,シフトを行います.これらの処理はアキュムレータのみでなくB,C,D,E,H,Lのすべてのレジスタについて実行することができます.

ビット操作命令は,8ビット・レジスタおよび,HLレジスタまたはIX,IYレジスタによって指定されたメモリの内容に対して,ビット単位でそのビットのON/OFFのチェック,セット/リセットを行います.

〈図2-11〉 実行順序の制御命令

(4) **実行順序の制御命令**

普通プログラムの命令は1命令ずつ,メモリに記憶されている順番に実行されていきます.この実行順序を変更することもできます.いくつかの命令を繰り返し実行する場合(ジャンプ)や,よく使う一連の命令などを一まとめにしてサブルーチンとして呼び出す(コール)場合などがあります(図2-11).

またプログラムの実行中,なんらかの条件によって次に実行するプログラムを変更したい場合があります.

これには,条件付きのジャンプ命令,または条件付きのコール命令があります.

ジャンプ命令とコール命令の違いは次のような点です.ジャンプ命令の場合は,対象となる番地へジャンプするのみでもどってくることは考慮されていません.

〈図2-12〉
相対ジャンプ命令JR *e*

*e*はジャンプ先を示す相対位置
ジャンプ先－ジャンプ元
*e*の出し方は，先に進む場合と前に戻る場合で異なる

(a) ジャンプ先がジャンプ元より先(上位の番地)にある場合

アドレス	機械語コード	ニモニック
5421	CB67	BIT 4, A
5423	2005	JR NZ, JMP
5425	3EFF	LD A, 0FFH
5427	32(nn)	LD (nnH), A
542A	DBF1	JMP: IN A, 0F1H

ジャンプ先を示すディスプレイスメント
*e*の機械語コードは(*e* －2)
e:　542A－5423＝07
e－2＝07－2　＝05

なぜ －2 するか？
相対ジャンプ命令は2バイト命令なので，この命令が読み込まれると，プログラム・カウンタが＋2されている．その分を補正するためである

(b) ジャンプ先がジャンプ元より前(下位の番地)にある場合

アドレス	機械語コード	ニモニック
312A	DBF3	LOOP: IN A, 0F3H
312C	CB6F	BIT 5, A
312E	28FA	JR Z, LOOP

ジャンプ先を示すディスプレイスメント*e*の機械語コードは|*e* －2|の2の補数
2の補数は，(1の補数＋1)
1の補数は，各ビットを反転させたもの
e：312A－312E＝－4
e－2：－4－2＝－6
|*e*－2|　：　0000　0110
1の補数：　1111　1001
2の補数：＋)　　　　　1
　　　　　　1111　1010
FA

用語解説
▶ディスプレイスメント：基準となるアドレスの位置と目的となるアドレスとの距離のこと

コール命令の場合は，所定のプログラムへジャンプする前に，コール命令の次に書かれている命令のアドレスを，スタックへ自動的に保存してからジャンプ先へジャンプします．

ジャンプ先での処理が終わったなら，RET命令で元のプログラムにもどります．これはRET命令によって，先程スタックへ保存したもどり番地がPCにセットされることで処理が行われます．このRET命令には条件付きの命令もあります．

8080Aのジャンプ命令は，ジャンプ先を絶対アドレスでしか表現できませんでした．Z80では自分自身のアドレスを中心に相対アドレスでジャンプ先を指定することができます(図2-12)．

この相対アドレス・ジャンプがあるため小さいプログラムなら，リロケータブルなプログラム(後述)をコーディングすることも可能です．

(5) CPU制御命令

割り込みの受け付けの可否，割り込みのモード設定，CPUの動作の停止命令もあります．

(6) 入出力命令

これだけは　知っておきたい

フラグによるデータのチェック

演算命令の実行結果の多くは，フラグ・レジスタにその結果が反映されます．フラグに反映される結果とは，

▶Zフラグ
　演算結果が0かどうか
▶キャリ
　キャリ，ボローによる大小のチェック
▶サイン・フラグ
　サイン・フラグAレジスタのD_7ビットがコピーされる
▶パリティ/オーバフロー
　論理演算結果のパリティを示し，オーバフローの有無が示される

このフラグの状況をプログラムに反映させるには，条件付きの分岐命令を用います．これによりフラグの状態に応じた分岐先が選択できます．

比較命令によって大小関係を調べる場合表2-Aのように Z, C, Sフラグによって行います．

〈表2-A〉
CP *m*の演算結果によるフラグの状態

演算結果 ＼ フラグの状態		Z	C	S(*)
A＝*m*		1	0	0
符号なし	A＞*m*	0	0	
	A＜*m*	0	1	
符号付き	A＞*m*	0		0
	A＜*m*	0		1

Aはアキュムレータの内容

(*)Sフラグは，符号付き数値を扱う場合のみ有効．アキュムレータのビット7で符号を表し，それがSフラグにコピーされている．
負のとき(M)　S＝1
正のとき(P)　S＝0

入出力命令は，入出力装置とのデータのやり取りを行う命令です．入出力のポートとして，OOHからFFHのアドレス・バスの下位8ビットで示されるアドレスが設定できます．

このアドレスの指定は，入出力命令の2番目のオペランドとして，直接アドレスを示す方法と，Cレジスタにアドレスをセットして示す方法があります．

(例)

```
IN    A, n
OUT   r, (C)
```

nは，1バイトのI/Oアドレス，rは，A,B,C,D,E,H,Lのレジスタを示す．

● フラグは判断を行うときのためにある

プログラミング時，しばしばなんらかの判断を必要とする場合があります．大小の判定，二つのデータが等しいかどうか，ビットごとでそのビットが1か0かをチェックしたり，演算結果が8ビットで表せなくなるときの桁あふれの有無など，多くの場面が考えられます．この処理にフラグ・レジスタが用いられます．フラグ・レジスタは8ビットのレジスタで，スタックへの処理などではAレジスタとペアで処理されます．そのときにはAFとニモニックで示されます(PUSH AF，POP AF)．

具体的には，表2-3に示すような命令の実行後，その命令の処理結果がフラグ・レジスタにセットされま

〈表2-3〉命令によるフラグ・ビットの変化

命令	D_7	D_6	D_4	D_2	D_1	D_0	フラグ・レジスタ内の対応するビット
	S	Z	H	P/V	N	C	
ADD A, s ; ADC A, s	*	*	*	V	0	*	8ビット加算，キャリを含む加算
SUB s ; SBC A, s ; CP s	*	*	*	V	1	*	8ビット減算，キャリ(ボロー)を含む減算，比較
NEG	*	*	*	V	1	*	アキュムレータの2の補数をとる
AND s	*	*	1	P	0	0	論理演算
OR s ; XOR s	*	*	0	P	0	0	
INC m	*	*	*	V	0	／	8ビット・インクリメント．16ビットのインクリメントはフラグは変化しない
DEC m	*	*	*	V	1	／	8ビット・デクリメント．16ビットのデクリメントはフラグは変化しない
ADD HL, ss; ADD IX, pp; ADD IY, rr	／	／	×	／	0	*	16ビット加算
ADC HL, ss	*	*	×	V	0	*	16ビット・キャリを含む加算
SBC HL, ss	*	*	×	V	1	*	16ビット・キャリ(ボロー)を含む減算
RLA ; RLCA ; RRA ; RRCA	／	／	0	／	0	*	ローテイト・アキュムレータ
RL m ; RLC m ; RR m ; RRC m SLA m ; SRA m ; SRL m	*	*	0	P	0	*	ローテイト，シフト
RLD ; RRD	*	*	0	P	0	／	アキュムレータと(HL)間の4ビットのシフト
DAA	*	*	*	P	／	*	アキュムレータの10進補正
CPL	／	／	1	／	1	／	アキュムレータの1の補数をとる
SCF	／	／	0	／	0	1	セット・キャリ・フラグ
CCF	／	／	×	／	0	*	キャリ・フラグの補数をとる
IN r, (C)	*	*	*	P	0	／	レジスタ間接入力
INI ; IND ; OUTI ; OUTD	×	*	×	×	1	／	ブロック入出力　B－1＝0の時Z＝1，ほかはZ＝0
INIR ; INDR ; OTIR ; OTDR	×	1	×	×	1	／	
LDI ; LDD	／	／	0	*	0	／	ブロック転送　BC－1≠0の時P/V＝1，ほかはP/V＝0
LDIR ; LDDR	／	／	0	0	0	／	
CPI ; CPIR ; CPD ; CPDR	*	*	*	*	1	／	ブロック・サーチ　A＝(HL)の時Z＝1，ほかはZ＝0 BC－1≠0の時P/V＝1，ほかはP/V＝0
LD A, I ; LD A, R	*	*	0	IFF	0	／	IFFの内容がP/Vにコピーされる
BIT b, m	×	*	1	×	0	／	Sのビットbの内容がZにコピーされる

(●プログラマが関与する必要のないフラグ)

S：サイン・フラグ
　演算の結果が負(D_7＝1)の時，S＝1
Z：ゼロ・フラグ
　演算の結果がゼロの時，Z＝1
●H：ハーフ・キャリ
　BCD演算結果の下位4ビットからのキャリ，ボローがあれば，H＝1
P/V：パリティ／オーバフロー・フラグ
　パリティ――奇数ならP/V＝1
　　　　　　偶数ならP/V＝0
　オーバフロー――オーバフローがあればP/V＝1
●N：加算／減算フラグ
　減算なら　N＝1
C：キャリ／リンク・フラグ
　アキュムレータの最上位ビットからの桁上がりでセットされる
　加算の時　キャリ｝があればC＝1
　減算の時　ボロー｝
　シフト，ローテイトによって，ビットのチェックを行える

*　操作の結果　変化する
／　操作の結果　変化しない
0　操作により，リセットされる
1　操作により　セットされる
×　不定
V　オーバフロー・フラグとして扱われる
P　パリティ・フラグとして扱われる
R　リフレッシュ・カウンタ
I　Iレジスタ（割り込みベクトルの上位バイト用）

す．それぞれの状態は，対応するビットを調べることで判定できます．しかし通常は，条件付きのジャンプ命令などを利用して判断しますので，フラグ・レジスタの中身まで立ち入ることはあまりありません．

実際の使用例をリスト2-1に示しておきます．

データの指定およびデータの格納されているアドレス指定方法〔アドレッシング〕

Z80のメモリは，16ビットで指定できる64Kバイトのアドレス空間が対象です．このメモリ中の特定のアドレスを指定する方法がいくつか用意されています．また，Z80の処理の対象となるデータは8ビットから16ビットで，これらのデータをなんらかの方法で指定する必要があります．以下に，データの指定およびデータの格納されているアドレスの指定方法について説明します．

(1) イミディエイト・アドレッシング

データの指定を直接命令の中に含めてしまう方法です．二番目のオペランドとして1バイトまたは2バイトの数値を指定します．機械語のコードは，命令コードに続いて1または2バイトの数値が続きます．したがって，最終的な機械語のコードは2ないし3バイト以上の大きさになります．

機械語のコードとなった場合の2バイトの数値データは，命令コードの次に下位バイト，その次に上位バイトの順番でメモリ中に格納されます．

(例)

```
LD  A, n
LD  A, 50 ; 3E50
            命令  オペランド
            コード
LD  BC, 5000 ; 010050
               命令  オペランド
               コード
```

また，プログラムの作成時に命令の一部としてデータを指定するため，定数の指定として利用できます．しかしプログラムの実行中に値の変化する変数データの指定には，以降に説明するデータの指定方法を用います．

(2) 拡張アドレッシング

ここで対象となるデータは，オペランドで指定されたアドレスのメモリの中身(内容)です．1バイトのデータの場合は，指定されたアドレスの中身そのものが対象となるデータです．2バイトのデータも，このアドレッシングで指定することができます．この場合，オペランドで指定されたアドレスの中身が2バイト・データの下位バイトとなります．指定されたアドレス+1のメモリの中身が2バイト・データの上位バイトとなります．

アセンブラの表記ではこの拡張アドレッシングの場合，

(nn)

と，2バイトのアドレスを示す数値を( )でくくって表示します．この( )は，この括弧でくくられた内容が示すアドレスのメモリの中身が対象となるデータで

あることを示します．

（例）

LD　HL，(5000H)

L ← メモリの5000番地の内容

H ← 5001番地の内容

2A 0050（命令コード／オペランド）

(3) レジスタ間接アドレッシング

汎用レジスタのペア・レジスタによって，16ビットのアドレスを指定します．レジスタの内容がメモリの特定のアドレスを指定します．その指定されたメモリの中身が対象となるデータとなります．

（例）

LD　A，(HL)；7E（命令コード）

対象となるデータは1バイトのデータのみです．2バイト以上のデータ処理の場合は，ペア・レジスタの内容を順次1ずつ加算していけば，連続したデータの処理も容易に行えます．

演算処理の可能なレジスタで，メモリのデータが指定できることでプログラミングの効率化が図られるようになります．

前記の拡張アドレッシングによって，書き換え可能なメモリ中のデータを処理の対象とすることができます．これにより演算結果，または途中経過などを保存しておける，変数の指定ができるようになります．

このレジスタ間接アドレッシングの機能には，次のような効果的な利用法があります．任意の変数に対して特定の処理を施す場合など，変数の指定をこのレジスタ間接アドレッシングで行います．この処理ルーチンを呼ぶ前に，所定の変数のアドレスをペア・レジスタにセットします．これにより，異なった変数に対する処理でありながら，まったく同一のプログラムで処理できます(リスト2-1参照)．

(4) 相対アドレッシング

Z80でメモリのアドレスを示す場合，64Kバイトのアドレス空間の特定のアドレスを絶対アドレスで示すのが基本となっています．したがって，アドレスの指定には16ビットのデータが必要となります．一方，ジャンプ命令ではジャンプ先がそのジャンプ命令のある番地の近辺であることが多くあります．

この場合，次のようなアドレスの指定の方法があります．そのジャンプ命令のある番地との相対的な位置を示すことで，ジャンプ先を指定できます．この相対アドレスを1バイトで表すと，絶対アドレスで表現するより1バイト分のプログラム・メモリを節約することができます(リスト2-1参照)．

この相対アドレスを用いることで，小さいプログラ

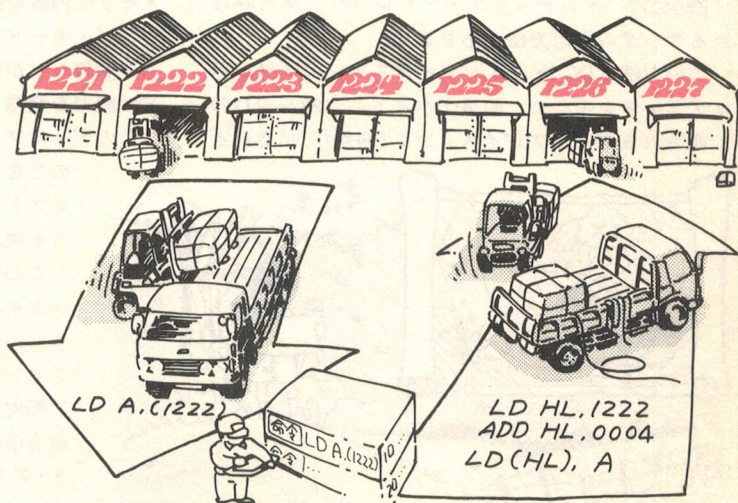
これだけは　知っておきたい

間接アドレッシングの表記法

アセンブラで書かれた命令のオペランドに(　)がついた場合は，次のような間接アドレスのモードになります．このモードでは，(　)の中の値をアドレスとするメモリ，I/Oポートの内容が処理対象となります．

この(　)の中には，数値の場合，メモリまたはI/Oデバイスのアドレスの値そのものが入ります．(　)の中がレジスタの場合は，レジスタの内容がメモリまたはI/Oデバイスのアドレスを示します．

いずれも処理の対象は(　)の中の値ではなく，(　)の中の値をアドレスとしたメモリ，I/Oデバイスの保持するデータが処理対象となります．

オペランドに(　)がない場合は，オペランドの値そのものが数値またはアドレスとして処理の対象になります．

ムならリロケータブルなプログラムとすることができます．そのプログラム内の絶対アドレスを参照していなければ，メモリ空間の任意の場所に置いても実行することができます．このようなプログラムをリロケータブルなプログラムといいます．これは80系の中でZ80でのみ可能でインテル社の8085A, 8080Aでは使用できません．

(5) インデックス・アドレッシング

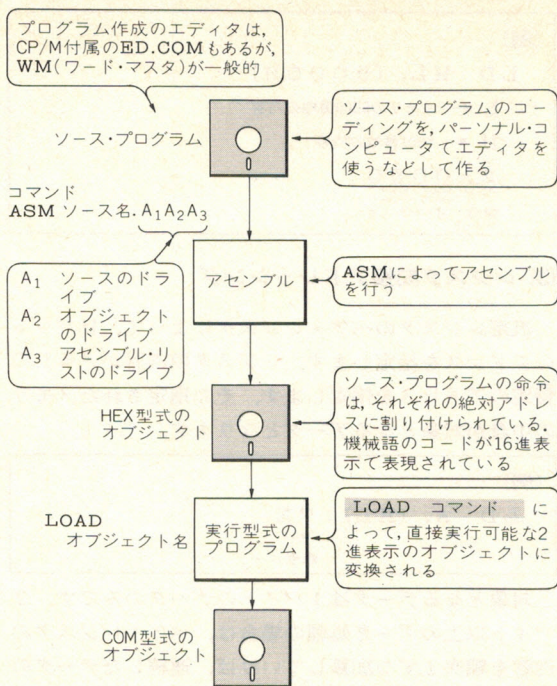
Z80は，インデックス・レジスタIX, IYをもっています．

このレジスタを用いて，インデックス・アドレッシングのモードが利用できます．IX, IYの16ビットのレジスタで，64Kバイトのメモリ空間の任意の場所が指定できます．このインデックス・レジスタで指定したアドレスとの相対位置を1バイトのオペランドとして，目的のアドレスを指定します．

レコードの先頭を示すアドレスをインデックス・レジスタで示すことで，複数のレコードを容易に処理することができます．

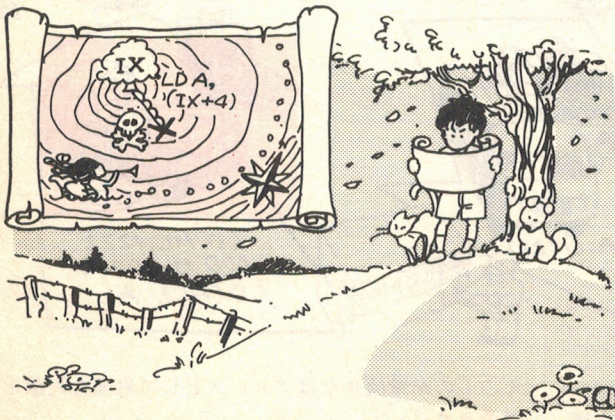
アセンブラでのプログラム開発

● **アセンブラは 機械語のコードを絶対アドレスに割り付けるものと，実アドレスへの割り付けをリンカに任せるものがある**

〈図2-13〉CP/MのASM(アセンブラ)によるプログラムの作成

アセンブラのソース・プログラムから，実行可能な機械語のプログラムが得られるまでには図2-13と図2-

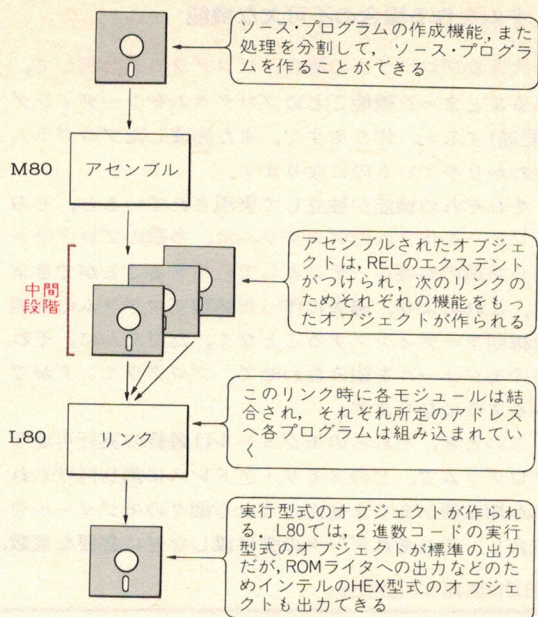
これだけは **インデックス・アドレッシングの使い方** 知っておきたい

Z80には，インデックス・アドレッシングと呼ばれるデータの指定方法があります．

この方法はIX, IYのインデックス・レジスタが示すアドレスを中心に，前後1バイト(−128〜＋127)で表されるディスプレイスメント(偏差)で示されるメモリの内容を処理の対象とします．このアドレッシング・モードの対象は，算術演算，論理演算，ビット演算などが可能で，テーブル・データの処理などに威力を発揮します．

このディスプレイスメントは，相対ジャンプ命令のときにジャンプ命令自身の置かれたアドレスと，ジャンプ先のアドレスとの差としても使用されます．

この相対アドレスとは現在地より5m前へとか，現在地より3歩後退などというように，ある基準からの相対的な位置を示すアドレッシング・モードです．

相対アドレス・ジャンプでは，ジャンプ命令の置かれているアドレス，インデックス・アドレッシングではIX，IYのインデックス・レジスタの示すアドレスが基準となります．

〈図2-14〉 M80などのリロケータブル・アセンブラによるプログラムの作成

〈図2-15〉 アセンブルの例（図2-14を参照）

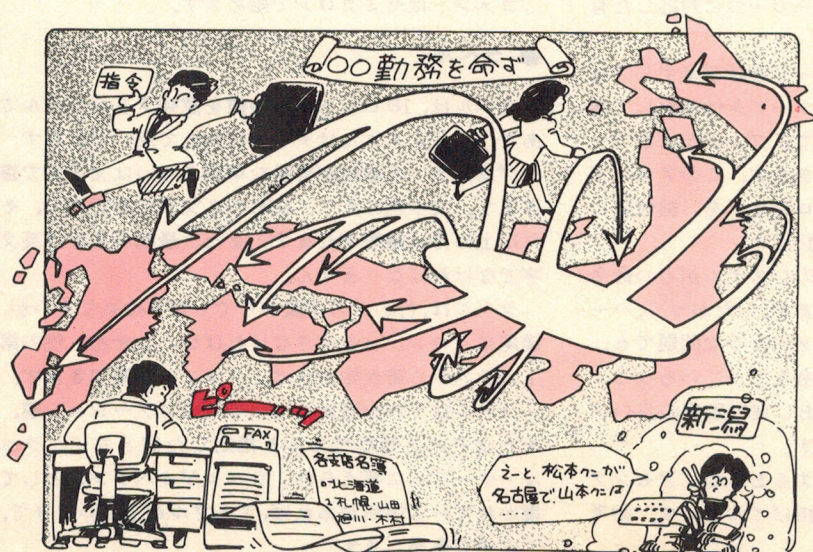
```
D>
D>A:WM Z80201.MAC ――エディタでソース・プログラムを作る．

D>A:M80 D:Z80201,D:Z80201=Z80201/L/R

No  Fatal error(s)              ソース・プログラムの
                                コンパイル

D>A:L80 Z80201,Z80201/N/E       リンカによるリンク作業

Link-80  3.43  14-Apr-81  Copyright (c) 1981 Microsoft

Data    0100    010F    <   15>   この作業で実行可能な
                                  Z80201.COMが作ら
41000 Bytes Free                  れる
[0100   010F        1]

D>A:ZSID Z80201.COM デバッガでオブジェクトの内容を調べる．
ZSID VERS 1.4
NEXT  PC  END          100番地からアセンブルされたオブジェ
0180 0100 A9FF         クトが入っている
#D100
0100: 3E 00 D3 0C 0E 01 CD 05 00 0E 00 CD 05 00 00 01
      >  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
0110: A6 0C 07 00 59 A0 50 00 38 00 CD 02 80 27 00 00
      .  .  .  .  Y  .  P  .  8  .  .  .  .  '  .  .
0120: 20 9E 1A 00 00 00 00 00 00 00 00 00 00 00 00 00
      .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
0130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
0140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
0150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
#L100
  0100  LD   A,00            逆アセンブルしてみる
  0102  OUT  0C,A
  0104  LD   C,01
  0106  CALL 0005
  0109  LD   C,00             ここまでがオブジェクト
  010B  CALL 0005 ―――┘
  010E  NOP
  010F  LD   BC,0CA6
  0112  RLCA
  0113  NOP
  0114  LD   E,C
#^C
D>
D>
```

14に示す二つの方法が代表的です．

図2-13の方法は，CP/M80のアセンブラで行われている方法で，アセンブラがソース・プログラムの情報に基づいて，機械語のプログラムが実行されるアドレスに直接割り付ける方法です．プログラムがどのメモリ・エリアで実行されるか，直接ソース・プログラムで指定できます．

そのため小さなプログラムでは，ソース・プログラムの記述の段階でどのように実行がなされていて，結果はどうだということが見通せます．ニモニック・コ

これだけは　知っておきたい

リンカによるリロケータブル・オブジェクトと実アドレスの決定

左の図は，大企業（大きなプログラム・システム）で人事異動が行われた様子を示している．異動が発令されて，支店（個々のプログラム・モジュール）に着いてはじめて，自分の立場や役職（絶対アドレス），同僚の配置（外部参照）などがわかる．

つまり，リロケータブル・オブジェクトという状態は配属されたところまでを表し，各支店は独立して機能しているが，リンカによって初めて，全社（最終の実行プログラム）が動き出すことを表している．

ードがわかれば，初心者でもソース・プログラムから容易に実行のようすがわかります．

図2-14で示した中間段階のオブジェクトは，実行可能な機械語のイメージのプログラムを16進数表示で表現したものです．実行可能なプログラムは二進数データで作られていて，ディスプレイの画面やプリンタに表示できません．

図2-14で示す処理は，ソース・プログラムではプログラムが完成した時点で，どのアドレスにセットされるかは関知せず，そのソース・プログラム内での関連だけを考慮して作成します．そして，それらソース・プログラムをアセンブルして作られたリロケータブルなオブジェクト・プログラムから，実際の実行時にメモリ領域を割り当てて完成させることを，リンカと呼ばれるプログラムに任せています．実行例を図2-15に示します．

この方法を用いるアセンブラの代表は，マイクロソフト社のM80と呼ばれるアセンブラです．

● リロケータブルなオブジェクトは大きなシステムを作る場合の不可欠な機能

大きなプログラムの場合，プログラムを分割して，あるまとまった機能ごとにプログラムをコーディング(記述)すると，作りやすく，また完成したプログラムもわかりやすいものになります．

それぞれの機能が独立して実現されていると，そのモジュール化されたプログラムは，多数のプログラムから共通の処理ルーチンとして利用することができます．これにより，過去に作った処理プログラムの機能を毎回コーディングすることなく，ただたんに，それらのモジュールを組み合わせて，プログラミングができるようになります．

このとき，それらのモジュールは最終の実行可能なプログラムで，どのメモリ・アドレスに割り付けられるか知る術がありません．これら個々のモジュールを統合し，それぞれデータの受け渡しなどに必要な変数，

アセンブラ

これだけは知っておきたい

コンピュータ・システムを動かす命令は，メモリに保存されています．そしてその命令は，1から数バイトの2進数，または16進表示で示されるコードで表されます．コンピュータの開発された初期の頃は，直接この機械語のコードでプログラミングしたそうです．今でも，デバッグの時など，16進表示のコードを直接メモリに書き込み，テストをする場合もあります．

しかし，直接この機械語のコードでプログラム全部を書くということは困難で，実質的に不可能なことです．そのために，機械語一つ一つに対応した有意な単語を割り当てます．

この単語をニモニック・コードと呼びます．プログラマは，このニモニック・コードを体系化して作られたアセンブリ言語を用いて，プログラムをコーディング(記述)することができます．このアセンブリ言語で書かれたソース・プログラムを，機械語に変換するのがアセンブラの役目です．

このアセンブリ言語は，そのシステムがもつ命令はすべて表現することができます．したがって，ハードウェア上の問題も含めて，どのような問題でも，システムがそれを処理する命令をもっているなら，100%そのCPUの機能を引き出し実行効率の良いプログラムを作ることができます．

以下に，アセンブラを利用するときの，アセンブラに対する指令の主なものを掲げておきます．参考にしてください．

● アセンブラの書式

ソース・プログラムは一定の書式で書く必要があります．その代表的な命令行を示します．

ラベル　　オペコード オペランド　コメント

ＬＯＯＰ：　ＬＤ　ＢＣ，ＭＡＸ；ＧＥＴ，ＭＡＸ

ここで，ラベルＬＯＯＰは，この命令行が置かれているアドレスを指すのに使うことができます．

オペランドとして，1ないし2個の語があり，これらは1ないしそれ以上のコンマ，またはスペースで区切ります．

コメントはセミコロンで始めます．

● ラベル

ラベルは，16ビットまでの値を意味するシンボルであり，アドレスまたはデータに代えて用いられます．

プログラマの識別が容易な単語あるいは文字列で書きます．文字列のうち，初めの6文字のみ有効で，それ以上の分は無視されます．また，最初の文字は英文字でなければなりません．

ラベルは，行のどの位置からでも書き始めてよいが，後尾にコロン(：)を付さなければなりません．行の第1カラムから書き始めれば，コロンは省略できます．

ラベルはそれが置かれている行のロケーション値，またはそれと等価であると指定した値と共に，ラベル・テーブルに蓄えられ，ラベルがオペランドとして書かれている箇所があれば，テーブルから引いてきて，その値を充当します．

ラベルの整合をとるのがリンカです(図2-16).

リンカが処理するためには，そのオブジェクトに次のような事項が含まれている必要があります．モジュール内のラベル，変数の相対アドレスおよびモジュール間での参照についての情報です．モジュールの配置先のメモリ・アドレスが決まったとき，この相対アドレスからすべての絶対アドレスが決まります．

参照のための情報とは，モジュール間で変数，ラベルを参照するため，参照元ではそれがほかのモジュールのものであることを示し，参照先ではそれがほかのモジュールからも参照されることを示します．それらのデータをもとに，リンカが結合処理を行います．このリロケータブル・オブジェクトの型式は，8ビット・システムではマイクロソフト社のリロケータブル・オブジェクトが標準となっています．

〈図2-16〉 リンカを使用するアセンブル
(各モジュール内のアドレスの参照はリンカの処理で行う)

このリンク作業で，プログラムをアセンブルした単

● 表現式

表現式は，1ないし数個の項から構成されていて，定数，変数，関数などが，演算子でつながれたものです．

この式は，アセンブラの解読能力によって種々の制限がありますが，Z80アセンブラでは，算術演算と論理演算の数種類が許されます．式は左から右へ順次計算され，括弧などで優先順位をつけることはできません．

演算子の種類を次に示します．

演算子	機能	演算子	機能
+	加算	/	除算
−	減算	&	論理積
*	乗算	!	論理和

スペースやタブのようなデリミタ(境界子，分離子)が式の中にあってもよいが，コンマは式の区切りを意味するので，使用の際には注意を要します．

演算は，すべて16ビットで行われます．8ビットの値しか許されない場合でも，演算結果が8ビット以内であれば，16ビットの値を含んだ式を使用してもかまいません．この場合下位8ビットだけが採用されます．

● 疑似命令(アセンブラ命令)

疑似命令は，Z80自身に対する通常の命令と異なり，アセンブラに対する命令です．

アセンブリ作業に指示を与えるだけで，命令そのものが，Z80の機械語に直接変換されることはありません．ただ，その命令のオペランドが機械語に変換されることはあります．例えば，DEFB命令などです．

アセンブラ命令を次に示します．

ORG　nn　　アドレスをnnにセットせよ．この命令によって，以降の命令群のメモリ上での位置が定められる．

EQU　nn　　この行のラベルの値をnnにし，プログラム内で，そのラベルがオペランドとして使われている所には，その値nnを充当せよ．

END　　ソース・プログラムの終端とせよ．この命令をソースの最後に付しておかないと，アセンブリ作業は正しく終了しない．

DEFB　n(DBとも書く)　　この行の位置(アドレス)に，値nをそのままセットせよ．

DEFB“S”(DB)　　この行の位置(アドレス)に，1文字Sのアスキ・コードをセットせよ．

DEFW　nn(DW)　　この行の位置(アドレス)と次の位置に，2バイトの値nnを，下位バイト，上位バイトの順にセットせよ．

DEFS　nn(DS)　　この行の位置(アドレス)から，nnバイト分だけ，メモリ領域を確保せよ．

DEFM“S”(DM)　　この行の位置(アドレス)から，文字列Sをアスキ・コードでセットしていけ．文字列Sの文字数は1から63までの範囲である．

(注)各行の位置は，アドレスを割り振っていくために，アセンブラ自身がもっているリファレンス・カウンタの内容に対応している．

位ごとのモジュールを結合し，任意の実行アドレスに配置することができます．このときプログラムを，命令部分とデータ部分に分離することができます．したがって，作成したプログラムをROM化することが容易に行えます．

また，各モジュール間での変数，ラベルの参照方法も，アセンブラで作成されるリロケータブル・オブジェクトだけでなく，各高級言語で出力されるオブジェクトも共通になっています．

マイクロソフト社のFORTRAN，Pascalなどの高級言語もこのリロケータブル・オブジェクトを作成するのは当然として，他社製のものとも互換性のあるオブジェクトを出力する機能をもっています．

アセンブラでハードウェアに密着した部分をコーディングし，データ処理などの部分を高級言語で作り，それぞれのリロケータブル・オブジェクトを，リンカで結合するなどということもできます(図2-17～図2-19)．

アセンブラの命令

アセンブラでプログラムを作るとき，機械語の命令に対応する命令コード以外に定数の設定，アセンブル作業の制御のための命令が用意されています(リスト2-2参照)．

このほかにもアセンブラのシステムによっては，マクロ機能(第９章に詳述)をもったものなど，システムの開発を容易にするための機能が追加されています．

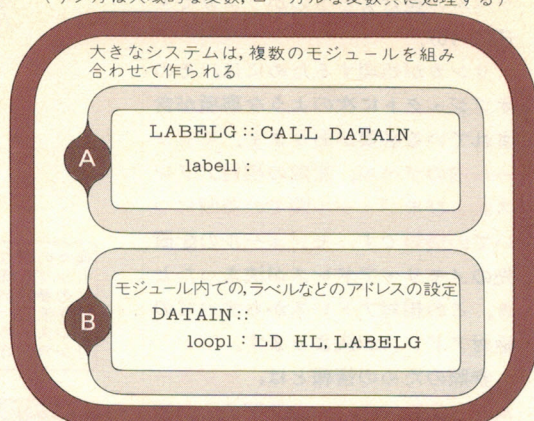
今回は，各アセンブラで共通なザイログ社型式のアセンブラの表記法について説明します．これで，アセンブラでプログラミングするための基本的な機能を修得することができます．その後，マイクロソフト社のM80のアセンブラのよく利用される機能を具体的な例を示して説明します(第９章参照)．

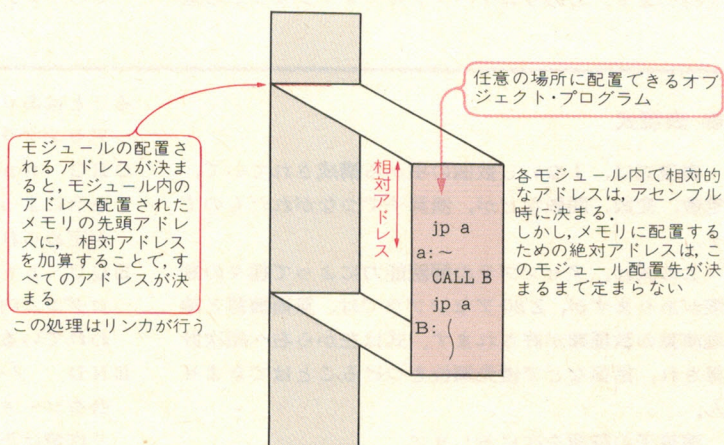
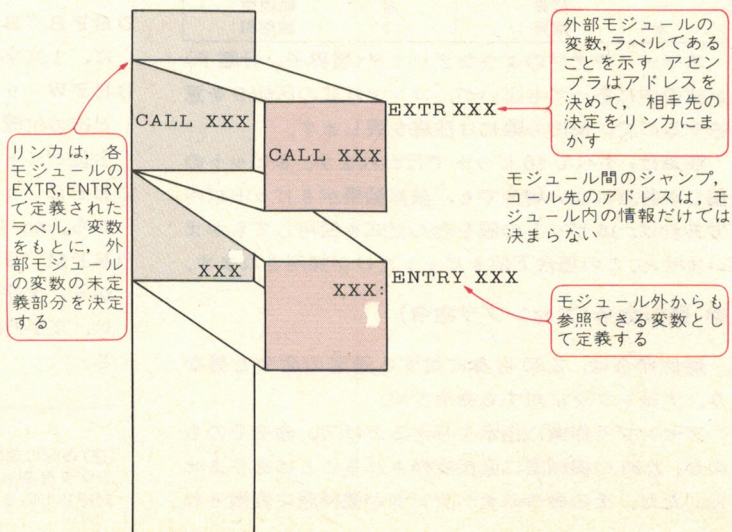
〈図2-17〉 モジュール化によるプログラム作成

(リンカは大域的な変数，ローカルな変数共に処理する)

大きなシステムは，複数のモジュールを組み合わせて作られる

Ⓐ
LABELG :: CALL DATAIN
labell :

Ⓑ
モジュール内での，ラベルなどのアドレスの設定
DATAIN ::
loop1 : LD HL, LABELG

＊システム全体で参照できるラベル(LABELG, DATAIN)
＊各プログラム・モジュール内のみで使用可能なラベル (labell, loop1)
＊Ⓐを高級言語で作り，Ⓑをアセンブラで作るようなこともできる

〈図2-18〉 リロケータブル・オブジェクトの配置

〈図2-19〉 モジュール間のジャンプ，コール

〈リスト2-2〉
アセンブル・リストの例
(プログラムそのものは特に意味のない処理を行っている

● アドレスの指定，定義に関する命令

アセンブラは，ソース・プログラムから機械語に変換したプログラムを，メモリの所定の場所(アドレス)に割り付ける作業を行います．このとき，どのアドレスに割り付けるのかを指定するのが，ORG命令です．

アセンブラは，このORG命令で指定されたメモリのアドレスから，順番に命令を設定していきます．設定していく途中で，特定の命令のセットされているアドレスを参照するときは，ラベルが使用されます．

このラベルは，分岐命令には不可欠な機能です．このラベルによってプログラムは，所定のアドレスを抽象化(ラベルの名前だけで)して参照することができます．ジャンプ先，参照するデータ・エリアなどを具体的なアドレスの数値で参照すると，プログラムの修正で参照アドレスがずれた場合など，それらの数値をすべて変更しなければなりません．

しかし，抽象化したラベルで指定しておくと，アセンブル時にアセンブラが最終的に確定したアドレスの値を割り当てます．したがってプログラムを修正するたびに，ラベルのアドレスを気にする必要がなくなります．アセンブルの終了を示すEND 命令は，必ずソース・プログラムの最後に必要です．このほかに定数を定義したり，ラベル，定数の有効範囲を指定するアセンブラ命令も用意されています(リスト2-2参照)．

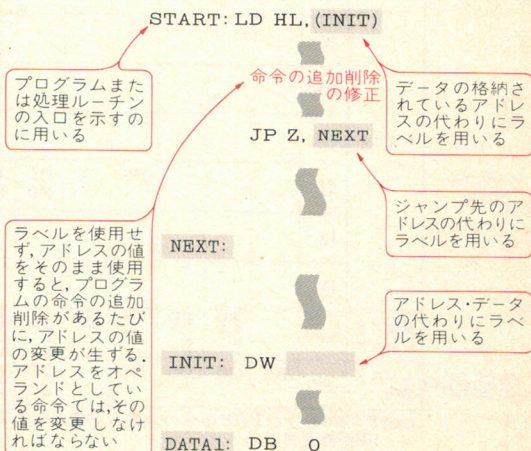
これだけは知っておきたい

ラベルを使用する効果

● 処理対象を抽象化できる

ラベルは記号番地とも呼ばれて，プログラムのアドレス・データを必要とする部分で，アドレスの値の代わりに用いられます．

実際のアドレスの割り当てはアセンブラが行いますので，プログラマはアドレスの絶対値についてほとんど考慮する必要はなく，プログラム作成が容易になります(図2-B参照)．

〈図2-B〉ラベルの使われ方

Z80命令コード表①

8ビット・ロード命令 "LD"

ソース側

		インプライド		レジスタ							レジスタ間接			インデックスド		拡張アドレッシング	イミディエイト
		I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+*d*)	(IY+*d*)	(*nn'*)	*n*
レジスタ	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E *d*	FD 7E *d*	3A *n'* *n*	3E *n*
	B			47	40	41	42	43	44	45	46			DD 46 *d*	FD 46 *d*		06 *n*
	C			4F	48	49	4A	4B	4C	4D	4E			DD 4E *d*	FD 4E *d*		0E *n*
	D			57	50	51	52	53	54	55	56			DD 56 *d*	FD 56 *d*		16 *n*
	E			5F	58	59	5A	5B	5C	5D	5E			DD 5E *d*	FD 5E *d*		1E *n*
	H			67	60	61	62	63	64	65	66			DD 66 *d*	FD 66 *d*		26 *n*
	L			6F	68	69	6A	6B	6C	6D	6E			DD 6E *d*	FD 6E *d*		2E *n*
レジスタ間接	(HL)			77	70	71	72	73	74	75							36 *n*
	(BC)			02													
	(DE)			12													
インデックスド	(IX+*d*)			DD 77 *d*	DD 70 *d*	DD 71 *d*	DD 72 *d*	DD 73 *d*	DD 74 *d*	DD 75 *d*							DD 36 *d* *n*
	(IY+*d*)			FD 77 *d*	FD 70 *d*	FD 71 *d*	FD 72 *d*	FD 73 *d*	FD 74 *d*	FD 75 *d*							FD 36 *d* *n*
拡張アドレシング	(*nn'*)			32 *n'* *n*													
インプライド	I			ED 47													
	R			ED 4F													

デスティネーション側

命令表記例　　機械語

(デスティネーション)(ソース)

- LD A, B　78
  - A←B
- LD A, (HL)　7E
  - A←(HL)
- LD A, (IX+03H)　DD 7E 03
  - A←(IX+03H)
- LD A, (1234H)　3A 34 12
  - A←(1234H)
- LD A, 12H　3E 12
  - A←12H
- LD (HL), A　77
  - (HL)←A
- LD (IX+05H), A　DD 77 05
  - (IX+05H)←A
- LD (1234H), A　32 34 12
  - (1234H)←A

16ビット・ロード命令, "LD", "PUSH", "POP"

ソース側

		レジスタ							拡張イミディエイト	拡張アドレッシング	レジスタ間接
		AF	BC	DE	HL	SP	IX	IY	*nn'*	(*nn'*)	(SP)
レジスタ	AF										F1
	BC								01 *n'* *n*	ED 4B *n'* *n*	C1
	DE								11 *n'* *n*	ED 5B *n'* *n*	D1
	HL								21 *n'* *n*	2A *n* *n*	E1
	SP				F9		DD F9	FD F9	31 *n'* *n*	ED 7B *n'* *n*	
	IX								DD 21 *n'* *n*	DD 2A *n'* *n*	DD E1
	IY								FD 21 *n'* *n*	FD 2A *n'* *n*	FD E1
拡張アドレッシング	(*nn'*)		ED 43 *n'* *n*	ED 53 *n'* *n*	22 *n'* *n*	ED 73 *n* *n*	DD 22 *n* *n*	FD 22 *n'* *n*			
レジスタ間接	(SP)	F5	C5	D5	E5		DD E5	FD E5			

デスティネーション側

PUSH命令→　　POP命令↑

(注) PUSHおよびPOP命令の全実行が終わったあとにSPが修正される

命令表記例　　機械語

(デスティネーション)(ソース)

- LD BC, 0659H　01 59 06
  - B←06H
  - C←59H
- LD BC, (0659H)　ED 4B 59 06
  - B←(0659H)
  - C←(065AH)
- LD (0659H), BC　ED 43 59 06
  - (0659H)←C
  - (065AH)←B
- PUSH AF　F5
  - SP←SP−1
  - LD (SP), A
  - SP←SP−1
  - LD (SP), F
- POP AF　F1
  - LD F, (SP)
  - SP←SP+1
  - LD A, (SP)
  - SP←SP+1

Z80命令コード表②

交換命令. "EX", "EXX"

		インプライド・アドレッシング				
		AF'	BC', DE' & HL'	HL	IX	IY
インプライド	AF	08				
	BC, DE & HL		D9			
	DE			EB		
レジスタ間接	(SP)			E3	DD E3	FD E3

命令表記例	機械語
EX AF, AF' AF↔AF'	08
EX DE, HL DE↔HL	EB
EXX BC↔BC' DE↔DE' HL↔HL'	D9

ブロック転送命令. "LDI", "LDIR", "LDD", "LDDR"

		ソース レジスタ間接 (HL)	
レジスタ間接	(DE)	ED A0	LDI
		ED B0	LDIR
		ED A8	LDD
		ED B8	LDDR

8ビット算術,論理演算命令. "ADD", "ADC", "SUB", "SBC", "AND", "XOR", "OR", "CP", "INC", "DEC"

INC, DECを除いてデスティネーションはすべてAレジスタであるが,ニモニックに表記するものとしないものがある

ソース側

	レジスタ・アドレッシング							レジスタ間接	インデックス		イミディエイト
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
ADD	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADC	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUB	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SBC	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
AND	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
XOR	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
OR	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
CP	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INC	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DEC	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

命令表記例	機械語
ADD A, B A←A+B	80
ADD A, (HL) A←A+(HL)	86
ADD A, (IX+07H) A←A+(IX+07H)	DD 86 07
ADD A, 33H A←A+33H	C6 33
ADC A, H A←A+H+キャリ	8C
SUB L A←A−L	95
SBC A, E A←A−E−キャリ	9E
AND (HL) A←A∧(HL)	A6
XOR C A←A⊕C	A9
OR B A←A∨B	B0
CP 2AH A−2AH	FE 2A
INC D D←D+1	14
DEC A A←A−1	3D

ブロック・サーチ命令. "CPI", "CPIR", "CPD", "CPDR"

レジスタ間接 (HL)	
ED A1	CPI
ED B1	CPIR
ED A9	CPD
ED B9	CPDR

▶HLはアキュムレータの内容と照合するメモリの位置のポインタ
▶BCはバイト・カウンタ

汎用算術演算命令. "DAA", "CPL", "NEG", "CCF", "SCF"

DAA	27
CPL	2F
NEG	ED 44
CCF	3F
SCF	37

16ビット算術演算命令. "ADD", "ADC", "SBC", "INC", "DEC"

ソース側 (縦: デスティネーション側)

		BC	DE	HL	SP	IX	IY
ADD	HL	09	19	29	39		
	IX	DD 09	DD 19		DD 39	DD 29	
	IY	FD 09	FD 19		FD 39		FD 29
ADC	HL	ED 4A	ED 5A	ED 6A	ED 7A		
SBC	HL	ED 42	ED 52	ED 62	ED 72		
INC		03	13	23	33	DD 23	FD 23
DEC		0B	1B	2B	3B	DD 2B	FD 2B

命令表記例 (デスティネーション, ソース)

命令表記例	機械語
ADD HL, BC HL←HL+BC	09
ADD IX, SP IX←IX+SP	DD 39
ADC HL, DE HL←HL+DE+キャリ	ED 5A
SBC HL, BC HL←HL−BC−キャリ	ED 42
INC SP SP←SP+1	33
DEC IX IX←IX−1	DD 2B

CPU制御命令. "NOP", "HALT", "DI", "EI", "IM"

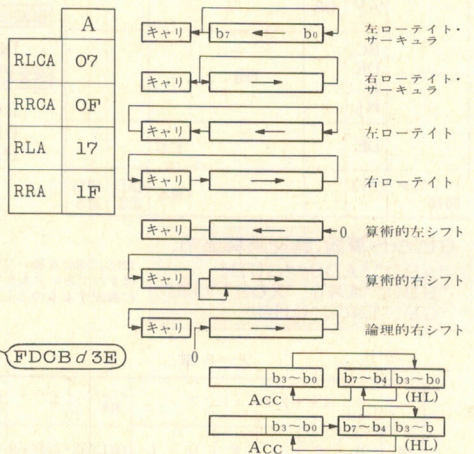
NOP	00	
HALT	76	
DI	F3	
EI	FB	
IM0	ED 46	モード0. 8080Aモード
IM1	ED 56	モード1. 38H番地へのコール命令
IM2	ED 5E	モード2. レジスタIと割り込みデバイスからの8ビット・データを用いた間接コール命令

Z80命令コード表③

ローテイト，シフト命令． "RLC", "RRC", "RL", "RR", "RLCA", "RRCA", "RLA", "RRA", "SLA", "SRA", "SRL", "RLD", "RRD"

ソースおよびデスティネーション

ローテイトあるいはシフトの型

	A	B	C	D	E	H	L	(HL)	(IX+*d*)	(IY+*d*)
RLC	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB *d* 06	FD CB *d* 06
RRC	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB *d* 0E	FD CB *d* 0E
RL	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB *d* 16	FD CB *d* 16
RR	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB *d* 1E	FD CB *d* 1E
SLA	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB *d* 26	FD CB *d* 26
SRA	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB *d* 2E	FD CB *d* 2E
SRL	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB *d* 3E	FD CB *d* 3E
RLD								ED 6F		
RRD								ED 67		

FDCB *d* 3E

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F

命令表記例	機械語
RLC B	CB 00
RRC C	CB 09
RL D	CB 12
RR E	CB 18

命令表記例	機械語
SLA H	CB 24
SRA L	CB 2D
SRL (HL)	CB 3E

ジャンプ，コール，リターン命令． "JP", "JR", "DJNZ", "CALL", "RET", "RETI", "RETN"

条件

			無条件	キャリ C	ノン・キャリ NC	ゼロ Z	ノン・ゼロ NZ	パリティ偶数 PE	パリティ奇数 PO	負 M	正 P	カウント
JP	拡張イミディエイト		C3 *n'* *n*	DA *n'* *n*	D2 *n'* *n*	CA *n'* *n*	C2 *n'* *n*	EA *n'* *n*	E2 *n'* *n*	FA *n'* *n*	F2 *n'* *n*	
JR	相対	PC+*e*	18 *e*−2	38 *e*−2	30 *e*−2	28 *e*−2	20 *e*−2					
JP	レジスタ間接	(HL)	E9									
JP	レジスタ間接	(IX)	DD E9									
JP	レジスタ間接	(IY)	FD E9									
CALL	拡張イミディエイト	*nn*	CD *n'* *n*	DC *n'* *n*	D4 *n'* *n*	CC *n'* *n*	C4 *n'* *n*	EC *n'* *n*	E4 *n'* *n*	FC *n'* *n*	F4 *n* *n*	
DJNZ	相対	PC+*e*										10 *e*−2
RET	レジスタ間接	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETI	レジスタ間接	(SP) (SP+1)	ED 4D									
RETN	レジスタ間接	(SP) (SP+1)	ED 45									

フラグによっては，一つ以上の目的で用いられる

命令表記例	機械語
JP 1600H PC←1600H	C3 00 16
JP Z, 1602H Z=1のとき PC←1602H Z=0のとき PC←PC+1	CA 02 16
JR LABEL+5 アセンブラでは，ディスプレイスメントではなく，ジャンプ先を示す数(数値，ラベル，式など)を記述する	18 03
JR C, LABEL−4	38 FA
JP (HL) PC←HL	E9
CALL 3344H $(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$ PC←3344H	CD 44 33
CALL NZ, 3400H Z=0のとき $(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$ PC←3400H Z=1のときPC←PC+1	C4 00 34
RET $PC_L \leftarrow (SP)$ $PC_H \leftarrow (SP+1)$ SP←SP+1	C9

Z80命令コード表④

ビット操作命令. "BIT", "RES", "SET"

	ビット	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
		レジスタ・アドレッシング							レジスタ間接	インデックス	
BIT	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RES	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

FDCB d A6

命令表記例	機械語
BIT 0, A	CB 47
Aレジスタの0ビットが, 0のとき Zフラグ=1 1のとき Zフラグ=0	
RES 3, (HL)	CB 9E
HLレジスタで示されるメモリの値の第3ビットを0にする	
SET 7, E	CB FB
Eレジスタの第7ビットを1にする	

リスタート命令. "RST"

	コール・アドレス	OPコード
コール・アドレス	0000H	C7
	0008H	CF
	0010H	D7
	0018H	DF
	0020H	E7
	0028H	EF
	0030H	F7
	0038H	FF

命令表記例	機械語
RST 00H	C7
$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$ $PC_H \leftarrow 00H$ $PC_L \leftarrow 00H$	
RST 38H	FF
$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$ $PC_H \leftarrow 00H$ $PC_L \leftarrow 38H$	

入力命令. "IN", "INI", "INIR", "IND", "INDR"

ソース 入力ポート

デスティネーション入力			イミディエイト (n)	レジスタ間接 (C)
IN	レジスタ・アドレッシング	A	DB n	ED 78
		B		ED 40
		C		ED 48
		D		ED 50
		E		ED 58
		H		ED 60
		L		ED 68
INI	レジスタ間接	(HL)		ED A2
INIR				ED B2
IND				ED AA
INDR				ED BA

(INI～INDR: ブロック入力コマンド)

命令表記例	機械語
IN A, (3EH)	DB 3E
入力ポート3EHからAレジスタへデータを入力する	
IN H, (C)	ED 60
Cレジスタで指定した入力ポートから, Hレジスタへデータを入力する	

出力命令. "OUT", "OUTI", "OTIR", "OUTD", "OTDR"

ソース側

	デスティネーション出力ポート		A	B	C	D	E	H	L	(HL)
			レジスタ							レジスタ間接
OUT	イミディエイト	(n)	D3 n							
	レジスタ間接	(C)	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69	
OUTI	レジスタ間接	(C)								ED A3
OTIR	レジスタ間接	(C)								ED B3
OUTD	レジスタ間接	(C)								ED AB
OTDR	レジスタ間接	(C)								ED BB

(OUTI～OTDR: ブロック出力コマンド)

命令表記例	機械語
OUT (06H), A	D3 06
出力ポート06HへAレジスタの値を出力する	
OUT (C), L	ED 69
Cレジスタで指定した出力ポートへ, レジスタの値を出力する	

レジスタの働き

● CPU内部にレジスタとよばれるデータを処理するためのエリアがある

データはメモリ中に用意されます．そして，それらのデータに演算処理を施し結果を得ます．その結果はメモリに得られるのではなく，アキュムレータと呼ばれるレジスタに得られます．また，処理の過程で一時的に途中の結果を保存しておくことが生じます．それらの処理を円滑に進めるためにも，レジスタと呼ばれるデータの処理エリアがCPUの内部に用意されています．

Z80の場合，このレジスタは図2-2に示すように，数種類のそれぞれの役割を与えられたレジスタが内蔵されています．

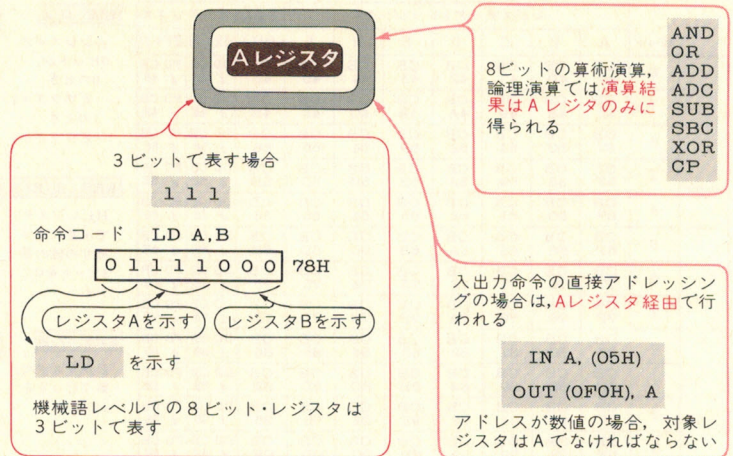
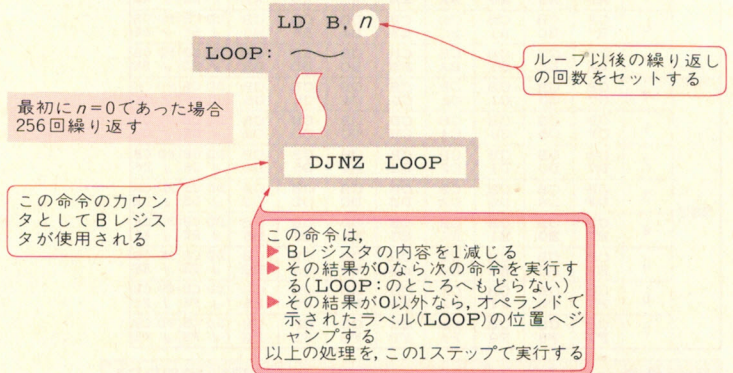
● データの演算は，8ビット，16ビットいずれの命令も用意されている

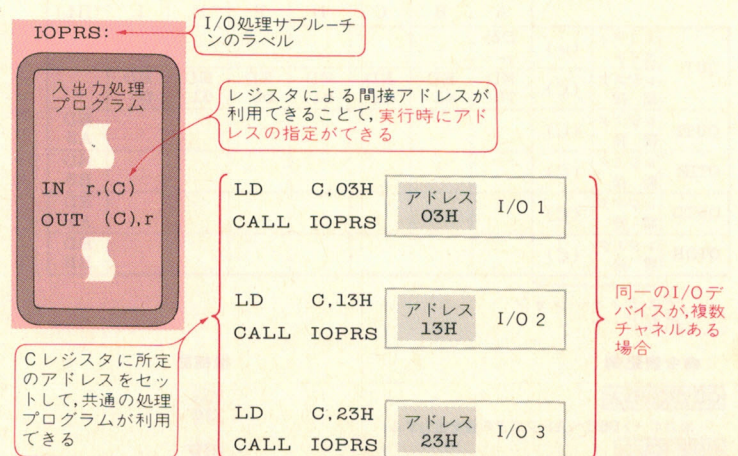
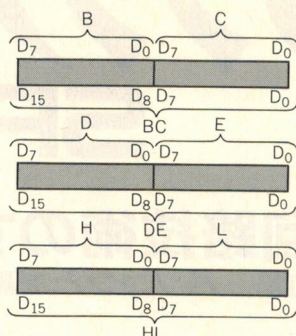
対称性(第2章Appendix①参照)にはいくぶん欠ける点がありますが，演算は8ビット演算,16ビット演算いずれも用意されています．なお，加減算は8ビット,16ビット共に用意されていますが，論理演算は8ビットの演算しかできません．演算結果のフラグに与える影響も様々ですので注意が必要です．

● Aレジスタは演算処理の中心となるレジスタでアキュムレータと呼ばれている

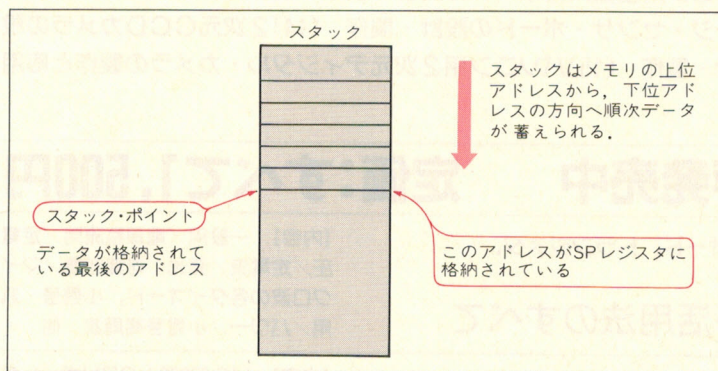
8ビット・データの演算の場合，その結果が得られるのはこのAレジスタのみです．また，直接アドレッシングによる入出力命令でも，このAレジスタ経由でなければなりません(図2-C)．

その他にも，対象がAレジスタでなければならない命令があります．割り込みベクトル・テーブルの上位アドレスなどは，Aレジスタからし

〈図2-C〉 Aレジスタはオールラウンド・プレイヤ

〈図2-D〉 Bレジスタは繰り返し実行する場合のカウンタとして利用される

〈図2-E〉 I/O処理のCレジスタによる間接アドレス指定

レジスタの働き

〈図2-F〉
8ビットCPUでもレジスタの組み合わせで16ビット処理が行える

〈図2-G〉 Z80の16ビット・レジスタ

SP　CALL, PUSH, POP命令などのスタック操作時の対象となる基準のアドレスを格納するレジスタ

IX,IY　Z80特有*の命令で，各種のメモリ・アドレッシングの基準となるIX, IYにメモリのアクセスする基準のアドレスをセットし，このインデックス・レジスタでアドレスを指定しメモリのアクセスを行う．オフセットを指定することもできる

＊8080A/8085Aにはない

かロードできません．レジスタ間接アドレスによってデータの転送を行う場合にAレジスタ以外は，データのアドレスを格納するレジスタに制限がつきます．

● Bレジスタは繰り返し処理のプログラム時の繰り返し数を示すカウンタになる

このBレジスタは，255回以下の繰り返しの場合のカウンタとして利用します．どのレジスタを使用しても繰り返し処理のプログラムを書くことはできます．

しかし，Z80ではDJNZ命令という便利な命令が用意されています．

この命令は，Bレジスタの値を1減じてその結果が0ならば次の命令に進み，もし0でなければオペランドで示されたラベルの位置にジャンプします．したがって，この命令のみで繰り返し処理が実現できます(図2-D)．

● Cレジスタは間接アドレス指定による入出力処理を実現する

8080AからZ80になって強化された機能で，このCレジスタで示されたアドレスのI/Oデバイスとの入出力処理が行えます．また，入出力の相手となるレジスタも，Aレジスタのみから，8ビットのレジスタすべてに拡張されています．

この機能の追加により，複数の入出力装置に対して共通な処理を行う場合，Cレジスタにそれぞれのアドレスをロードして，共通なサブルーチンをコールすることで容易に実現できるようになりました(図2-E)．

● 8ビットのレジスタを対にして16ビットのレジスタとして使用できる

8ビットの各レジスタは，それぞれ次のように組み合わせて16ビット・レジスタとして使用します．BC，DE，HLがそれぞれ対となるレジスタです．その中でHLのペア・レジスタは，16ビット・レジスタのアキュムレータの役割を与えられています．また，この16ビット・レジスタ間の加減算は用意されていますが，ディクリメント，インクリメント命令などではフラグが変化しないなどと，8ビットの命令と異なっている部分がありますので注意が必要です(図2-F)．

その他にZ80は，16ビットのインデックス・レジスタをもっていて，メモリ中のデータを指定するアドレッシングを多様なものにしています(図2-G)．

システム構成の基本

■ NEXT

CPUとメモリ/ペリフェラルの接続の仕方，データの読み書きのタイミングをどのように設計するかを，具体的に示しながら説明します．

keywords

アドレス・デコーダ：指定されたアドレスで，それぞれの素子をイネーブルする回路．

ペリフェラル：周辺の意．CPUの周囲にあって，各種の機能を実現するデバイスを指す．

DC特性　：直流特性．入出力電圧レベル，駆動電流の量などの特性．

AC特性　：処理速度に関する特性．出力を指定してからデータ確定までの時間など，タイミング設計の基礎となる．

NMOS：電子をキャリアとしたMOSトランジスタ．多くのLSIがこのNMOS．

C-MOS　：P，Nチャネルの異なったMOSトランジスタを組み合わせて作る．低電力化，高速化が図れるため，多くのデバイスがC-MOS 化されている．

マシン・サイクル：CPUの処理の最小単位．命令は1から数個のマシン・サイクルで構成される．中には数十になるものもある．

M80　：マイクロソフト社製のZ80，8080A用のマクロ・アセンブラ．

この章では，Z80のCPUチップを使用し，最小限の機能をもったシングル・ボード・コンピュータ・システムを示し，その概要について説明します．Z80のシステムの全体像を把握し，各コンポーネントの組み合わせ，データの大きな流れをつかむことを目的とします．

その次に，Z80 CPUの各端子の説明を行います．

シングル・ボード・コンピュータを構成する基本的な機能

Z80を使用した最小システムとしては図3-1に示すように，CPUチップ，ROM，専用のパラレルI/O用のLSIと数個のTTLで構成することができます．この場合，RAMをもっていないためプログラム作成上多くの制約が生じます．しかし，データを受け取り，なんらかの加工をし結果を再び出力するような処理では，十分に実用になります．

Z80は，内部に多くのレジスタをもっているので，データ保存のバッファ代わりに利用できます．14バイトくらいはとれそうです．

次にRAMを追加した小規模なシステムを考えます．RAMを追加することで，かなり大量のデータをRAMのメモリ上に一時的に保存することができます．またプログラムの作成上スタックが使え，サブルーチンがコーディングできるようになります．これは，プログラムの作成上必要不可欠なことです．

前記のRAMなしのシステムは特殊な例で，一般的には図3-2のシステムが最小システムと考えられます．CPUチップ，TTLによるクロック発振部，RAM，ROMによるメモリ部，シリアル，パラレルのI/Oペリフェラル LSIによる入出力部から構成されています．

最近は，大容量のROM，RAMが容易に利用できますので，このボードでも16Kバイト ROM，8Kバイト RAMのそれぞれ1個のメモリ・チップを使用して実現できます．

このように，小規模なシステムの場合，バスに接続される素子の数が少なく，CPUの出力端子のドライブ能力だけで各素子を十分駆動することができます．したがって，バッファなどの回路が省略でき，非常にシンプルなシステムになります．

また，アドレス・デコーダについても，多くの場合16ビット（メモリ），8ビット（I/O）のアドレス・ラインをすべて使用するわけではありません．そのシステムで使用する各素子のみについて識別できれば，それで十分アドレス・デコーダの機能を果たします（図3-3）．

しかしこの場合，同一のメモリ・チップに対して複数以上のメモリ・エリアが割り付けられることになり

〈図3-1〉 Z80ミニマム・システムの構成例

パワーオン・リセット．電源投入時自動的にリセットがかかる

メモリ素子が一つしかないので，メモリの$\overline{CS}$に直接接続できる

A_{15}～A_{13}は接続していないので，メモリのアクセスがあったとき，メモリ空間を0000～1FFFHの8バイトごとに区切ったいずれの領域としても指定できる．
しかし，プログラムのスタートは0000Hからであるので，0000～1FFFHとしてプログラムする

2764(64Kビット ROM)

Z80

リセット・スイッチ　1S1588　5V　100k　1k　10μ

これらの使用しない信号は5Vに接続しておく

アドレス・バス全体で16ビットあるが，すべてを使用する必要はない

データ・バス8ビット

データの入出力は，データ・バス経由で行われる

Z80 PIO　ポートA　ポートB

Z80ファミリなのでCPUと直結できる端子が多い

プログラムによってこれらの端子の役割を設定する

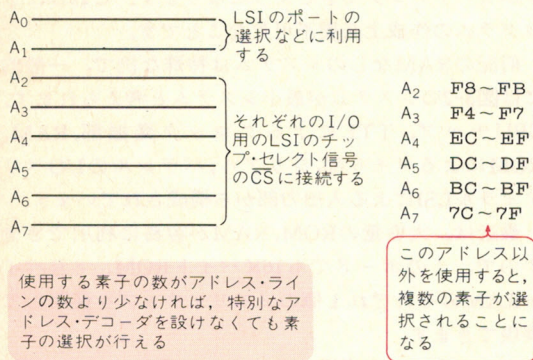
Z80ファミリの周辺用LSI．詳細は第5章～第7章で説明する

内部レジスタを選択するためにアドレス・バスのA_0とA_1をつなぐ

“H”レベルを確保するためのプルアップ抵抗

560Ω　1.8k　220Ω　LS04　220Ω　LS04　LS04　300Ω　5V　2MHz

〈図3-3〉 デコーダの回路を設けずI/O素子を選択する場合

接続するアドレス・ライン	いずれか1本をI/Oデバイスの$\overline{CS}$に接続する						I/Oデバイスのレジスタ・ポート選択端子に接続		各ポートのアドレス
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
A_2	1	1	1	1	1	0	0	0	F8H
							0	1	F9H
							1	0	FAH
							1	1	FBH
A_3	1	1	1	1	0	1	0	0	F4H
							0	1	F5H
							1	0	F6H
							1	1	F7H
A_4	1	1	1	0	1	1	0	0	ECH
							0	1	EDH
							1	0	EEH
							1	1	EFH
A_5	1	1	0	1	1	1	0	0	DCH
							0	1	DDH
							1	0	DEH
							1	1	DFH
A_6	1	0	1	1	1	1	0	0	BCH
							0	1	BDH
							1	0	BEH
							1	1	BFH
A_7	0	1	1	1	1	1	0	0	7CH
							0	1	7DH
							1	0	7EH
							1	1	7FH

ます．そのことを考慮してプログラムを作成すればよいのですから，なんら問題は生じません．

汎用のボード・システムなどの場合に使われるアドレス・デコーダは，すべてのアドレス・ラインを使用してデコードしておく必要があります．そうすれば，ボードを追加したときに，アドレスの指定で混乱することが避けられます．

Z80の出力端子のドライブ能力および入力端子の特性

Z80の各端子のDC特性を図3-4に示します．DC特性とは，“H”レベル，“L”レベルのそれぞれの範囲，入力電流，出力駆動能力などの仕様です．タイミング，データの伝達特性などは，AC特性として個々の周辺装置との接続法について説明するときに具体的に述べ

〈図3-2〉小規模なZ80のシステム

メモリのアクセスにはRD, WRだけではだめ．MREQの信号のでている間のタイミングを考えてANDをとる

メモリ専用の書き込み信号

メモリ専用のリード信号

メモリの読み出し

メモリ・マップ

	A_{15}	A_{14}	A_{13}
RAM	1	0	×

8000～9FFFH
A000～BFFFH
のいずれの範囲でも利用できる

	A_{15}	A_{14}	A_{13}
ROM	0	×	×

0000～1FFFH
2000～3FFFH
4000～5FFFH
6000～7FFFH
このいずれの範囲でもアクセスできるが，0000～1FFFHとしてプログラムをコーディングする

単位はボー（BPS）ボーレート選択

コントロール・バスの入力端子は抵抗でプルアップしておく．後で使用するときは，そのまま入力信号を接続するだけでよい

I/O用のRD信号

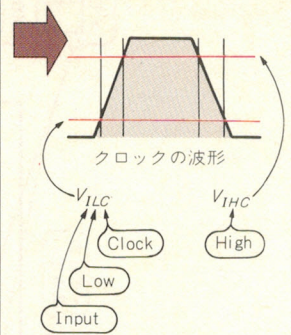
I/O用のWR信号

I/Oをアクセスするときには$\overline{RD}$, $\overline{WR}$信号と$\overline{IORQ}$のタイミングを考えてANDをとる

プログラマブル・ペリフェラル・インターフェース．使用目的に応じて多様なモードが設定できる．5章，6章で説明する

〈図3-4〉 Z80 CPUのDC特性

(T_a=0°C～+70°C, V_{CC}=+5V±5%)

記号	項目	最小値	最大値	単位	測定条件
V_{ILC}	クロック"L"入力電圧	−0.3	0.45	V	
V_{IHC}	クロック"H"入力電圧	V_{CC}−0.6	V_{CC}+0.3	V	
V_{IL}	"L"入力電圧	−0.3	0.8	V	
V_{IH}	"H"入力電圧	2.0	V_{CC}	V	
V_{OL}	"L"出力電圧		0.4	V	I_{OL}=1.8mA
V_{OH}	"H"出力電圧	2.4		V	I_{OH}=−250μA
I_{CC}	消費電流		150	mA	
I_{LI}	入力リーク電流		10	μA	V_{IN}=0～V_{CC}
I_{LOH}	3ステート出力リーク電流		10	μA	V_{OUT}=2.4V～V_{CC}
I_{LOL}	3ステート出力リーク電流		−10	μA	V_{OUT}=0.4V
I_{LD}	入力時のデータ・バスのリーク電流		±10	μA	0≦V_{IN}≦V_{CC}

▶クロック端子はTTL仕様になっていない．したがってTTLのクロック・ジェネレータからクロックを得るとき，少なくとも300Ωくらいの抵抗でV_{CC}にプルアップする．

▶各端子の出力ドライブ電流はLS TTL 4個分の能力しかない．大きなシステムでは各バスにバッファが必要となる

〈表3-1〉 TTLの標準入出力特性(1入力端子)

項目／シリーズ	等価的な入力特性 (kΩ)	"H"時の最大入力電流 I_{IH}(μA)	"L"時の最大入力電流 I_{IL}(mA)	ゲート当たりの伝達時間 (ns)	ゲート当たりの消費電力 (mW)	処理できる周波数 (MHz)	"H"時の最大出力電流 I_{OH}(μA)	"L"時の最大出力電流 I_{OL}(mA)
54/74	4	40	−1.6	10	10	DC～35	−400	16
54H/74H	2.8	50	−2	6	22	DC～50	−500	20
54L	40	10	−0.18	33	1	DC～3	−100	2
	8	20	−0.8					
54LS/74LS	18	20	−0.4	9.5	2	DC～45	−400	4/8
54S/74S	2.8	50	−2				−1000	20
54ALS/74ALS	40	20	−0.2	4	1	DC～100	−400	4/8
54AS/74AS	2.7	200	−2	1.5	22	DC～200	−2000	20
フェアチャイルド 74Fシリーズ		20	−0.6	3.0	4	DC～100	−1000	20

*最近は，各メーカからHCシリーズと呼ばれるC-MOSを使うことも多くなっている．これらは，LS TTLと同等な処理速度をもち，ドライブ能力も従来のC-MOSがLS TTLを1素子しかドライブできなかったのに対し，LS TTLを10素子までドライブすることができる．このため，LS TTLと混在させることが可能になる．また，さらに高速のACシリーズも順次そろっている．

*ただし，HCシリーズをドライブする場合は，"H"レベルを確保するために，3.3kΩくらいのプルアップ抵抗が必要．この部分のみコンパチブルでなくなるが，その他は，LS TTLと同様に扱える．

*54は74シリーズのMILスペック品

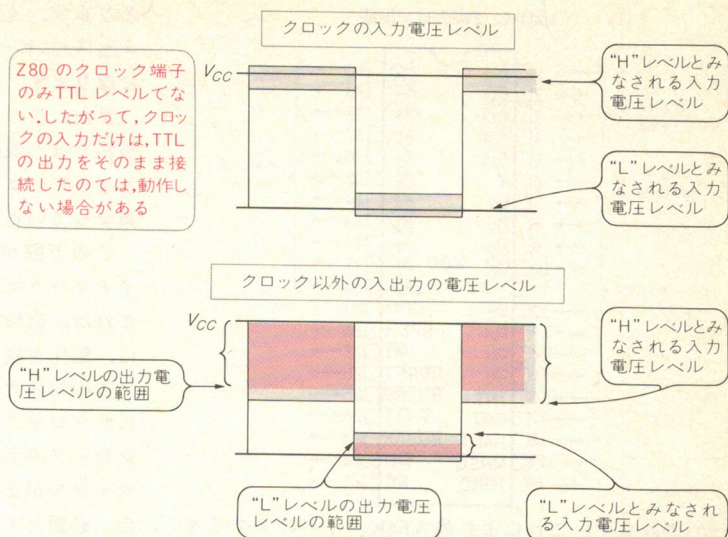
ます．

TTLから，NMOSまたはC-MOSをドライブする場合は，入力インピーダンスが非常に高いのでドライブ電流の問題はありませんが，入力容量が比較的大きいため，容量性負荷の増大によるスピードの低下の問題が生じます．

NMOS，C-MOSからTTLをドライブする場合，レベルの問題は生じませんが，"L"レベルのときの吸い込み電流の容量が少ないので，せいぜいTTLを一つドライブするだけの能力しかありません．なお，表3-1にTTLの入出力特性を示します．

これらそれぞれの素子のドライブ能力はデータ・シートにI_{OL}("L"レベル出力電流)，I_{OH}("H"レベル出力電流)，そのときの出力電圧の"L"レベル，"H"レベルが載っています．この出力特性と相手側の入力特性がマッチするかどうかのチェックを行います(図3-5)．

図3-4のZ80CPUチップの各出力端子のドライブ能力をもとにして，そのシステムで必要とする各バスのドライブ能力を満足するかどうかチェックします．不足する場合は，バッファを設けてドライブ能力を強化します．

〈図3-5〉 Z80の入出力電圧特性

Z80の各端子の機能についての概要の説明

● データ・バス(双方向性のバス)

データ•バスは, 8本あります(図3-6). この8本が8ビットのデータを表し, これにより8ビットCPUと呼ばれています. CPUチップは, データを読み込むと共に, 外部にデータを書き出すことも行います. そのためデータ・バス上の信号は, 一方通行でなく両方向へデータが行き来します.

しかしこのデータの移動は, 二車線の道路を同時にデータが行き来するわけではありません. 8ビットのデータの送受信を8本の信号線で行うのですから, 単線の線路を交互に運転されるローカル線みたいなものです. このポイントの切り替えに相当する信号がCPUから出ていて, データ・バス上のバッファの方向の切り替えに利用されています.

● アドレス・バス

16本のアドレス・バスがZ80には用意されています. この16本のアドレス線で, 16ビットのアドレス信号が

これだけは　**ファンアウト**　知っておきたい

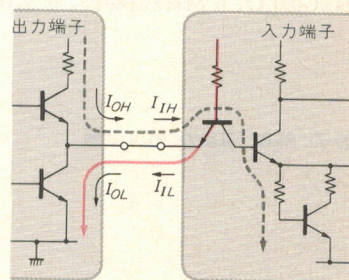
TTLは, 入力出力が図3-Aのようになっています. したがって, 入力端子を"L"の状態にすると, 入力段のトランジスタのエミッタ電流が流れることになります. この電流の値は, TTLの各タイプによって異なります.

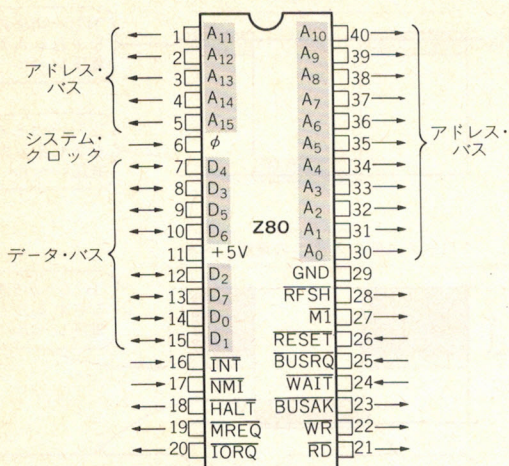
高速タイプの74S, 74ASでは2mAが最大で, 現在一般に使用されているショットキの74LSタイプは0.4mAとなっています. スタンダードのTTLでは, 1.6mAが入力端子を"L"の状態にするために, 入力端子より吸い込む必要のある電流の最大値です.

"H"の入力に対しては, 逆バイアスになるため, 20μA〜40μAの電流しか流れ込めません.

一方, TTLの出力特性は入力特性に応じて, "L"のときは相手から8mA〜16mAの電流を吸い込むことができるようになっています.

これら入力と出力間の電流の比がファンアウトとなります. そしてファンアウトは, その出力端子がいくつの入力端子を駆動することができるかを示します.

〈図3-A〉
TTL同士の接続

具体的には74LSの素子は$I_{IL}=-0.4$mA, $I_{OL}=8$mAなので, ファンアウト$=I_{OL}/I_{IL}=8/0.4=20$となり, 最大20個の74LSタイプをドライブすることができる

〈図3-6〉 Z80のピン配置

出力されます．これにより最大64Kバイトのメモリを制御することができます．64Kバイトは，10進数の普通の表現では65536バイトとなります．

メモリの読み書きのときは16本のアドレス線がすべて利用されます．しかし，入出力装置との間でデータの交換を行うための入出力命令では，アドレス線は下位の８本しか使用しません．上位のアドレス・ラインにはAレジスタの内容が出力されます．したがって，通常は入出力装置を最大256個までしか割り当てられません．

コントロール・バスの個々の端子についての説明

Z80は，CPUの動作をコントロールするための制御信号を出したり，外部の状況を検出する端子によるコントロール・バスをもっています．

● システム・クロック

このクロックは，マイクロコンピュータ・システムの動作の基準になります．CPUチップのグレードによってクロック周波数の上限が2.5MHz(Z80)，4 MHz(Z80A)，6 MHz(Z80B)，8 MHz(Z80H)と各種あります．4 MHz以上の高速のバージョンでは，アクセス・スピードの遅いROMの接続などで工夫が必要になります．

また，このシステム・クロックには下限があります．規格では下限のシステム・クロックの周波数は500kHzとなっています．この値は，上限のクロック周波数の異なっているものでも共通です．

この下限があるということは，CPUチップ内にダイナミックな動作を必要とする部分があるためです．これは，自転車が走り続けなければ倒れてしまうように，動作を続けなければ状態を維持できないからです．

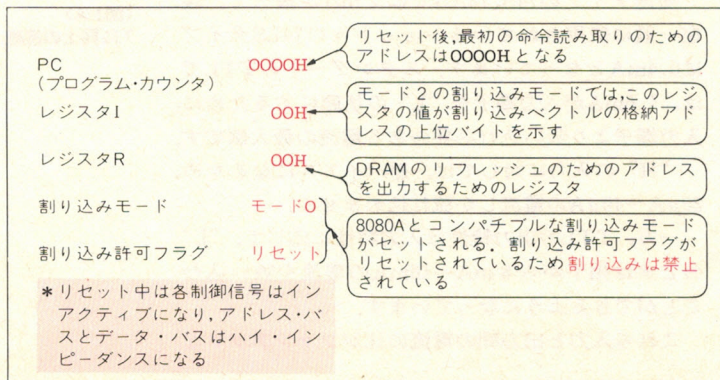
このシステム・クロックは，TTLの発振回路で作成したクロックを使用することができます．システム・クロックの上限付近で使用する場合は，デューティ・サイクルが正確に50％でなければなりません．この場合，必要とするクロックの倍のクロックを作り，それを1/2に分周することでシステム・クロックとします．

また，このシステム・クロックの入力端子はほかの端子と異なり，“H”レベルでの入力電圧がTTLとコンパチブルでなく，$V_{CC}-0.6$の4.4Vが必要です．このために300Ωくらいの抵抗で電源にプルアップしておきます．さらに，クロックの立ち上がり，立ち下がりの時間にも制限があり，30ns以内でなければなりません．

● リセット(システムの最初の状態)

リセット端子は，通常は“H”の状態にしておきます．この端子を“L”にするとZ80は，初期化されます．初期化された状態は，図3-7に示すようにPC(プログラム・カウンタ)は０００００H，すなわちゼロ番地からのプログラムを次に実行するようにセットされます．

このリセット信号は，CPUをリセット(入力)するだけでなく，このシステムに接続されている，各I/Oコントロール用のLSIもリセットできるようにしておきます．この場合，周辺用のLSIのリセット入力が“H”でアクティブなものもあるので注意が必要です．

〈図3-7〉 RESET直後のCPUの状態

また，リセット信号によって素子が初期化するためのリセット信号のアクティブの期間が，それぞれの素子について別に定められています．

さらに，水晶発振の振幅安定にも時間が少しかかります．その時間が，CPUの必要とする期間より長い場合があります．それは，データ・シートで調べることができます．できたら確認してみてください．一般に100〜500msぐらいの値が使われます．

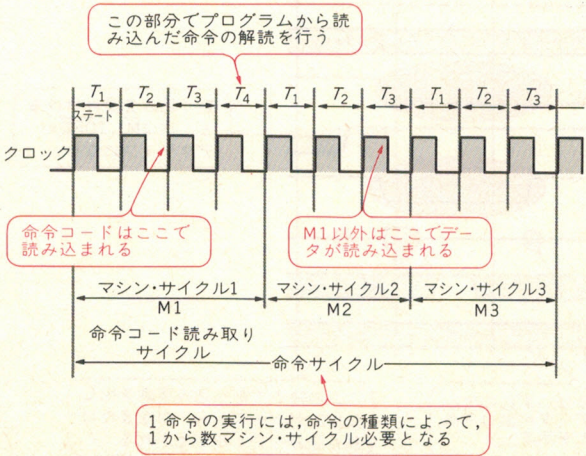
Z80のCPUチップは，DRAMのリフレッシュのコントロールを直接行っています．RAM上のデータを保存するために，リセットについてもいくつかの留意すべき事項があります．これについては，DRAMの説明の所でふれます．

● $\overline{\mathrm{M1}}$(エム・ワン・サイクル)

Z80の外部とのデータ(プログラム，データ)のやりとりは，図3-8に示すように，システム・クロックの1サイクルを1ステートとして行われます．そして，Z80の意味ある処理の最小単位を1マシン・サイクルといいます．

このマシン・サイクルは，三つないし四つのステートより構成されます．各マシン・サイクルは，CPUと外部の入出力，演算の最小単位となります．一つの命令の実行は，その命令の種類に応じて，1から数マシン・サイクルで処理されます．

そしてプログラムの処理には，必ず最初に実行する命令のコードを読み込みます．これは，命令サイクルの最初のマシン・サイクルで，このサイクルに，命令コードの解読を行い，特別にM1サイクルと呼ばれます．

このM1サイクルは，割り込み，ダイナミック・メモリのリフレッシュのために，特別な意味をもっています．この$\overline{\mathrm{M1}}$端子が“L”のとき，CPUがM1サイクルを実行中であることを示します．このM1サイクル時のみ4ステートで構成されています．そしてT_4ステートのときは，読み込んだ命令コードの解読作業を行います．この解読作業中CPUは，外部との接触がありません．このM1サイクルのT_4ステートでダイナミック・メモリのリフレッシュが行われます．

〈図3-8〉 Z80 CPUの基本動作

● リード/ライト動作時に必要となる端子についての説明

メモリ，入出力装置とCPUとの間でデータの交換を行うための信号として，次の四つのラインが用意されています．

▶$\overline{\mathrm{RD}}$(ReaD)：CPUが CPU外のメモリ，入出力装置からデータを読み込むときにアクティブになる．

▶$\overline{\mathrm{WR}}$(WRite)：CPUが CPU外のメモリ，入出力装置にデータを書き込むときにこの信号がアクティブになる．

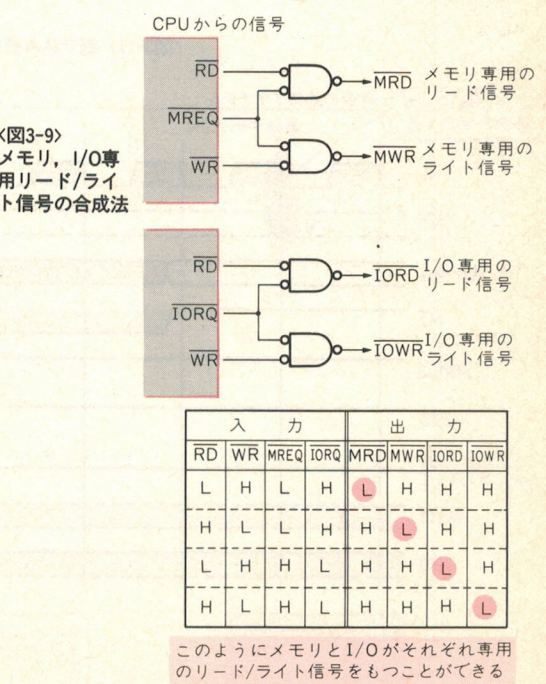
▶$\overline{\mathrm{IORQ}}$(I/O ReQuest)：入出力の対象がメモリでなくIN/OUT命令によって読み書きされる入出力装置であることを示す．割り込み処理時にベクタをCPUが読み込むときにも，この信号がアクティブになる．

▶$\overline{\mathrm{MREQ}}$(Memory REQuest)：入出力の対象がメモリであることを示す信号．

これら四つの信号の組み合わせで，メモリ，入出力装置，それぞれ専用のリード/ライト信号を作ることができます(図3-9)．

入力				出力			
$\overline{\mathrm{RD}}$	$\overline{\mathrm{WR}}$	$\overline{\mathrm{MREQ}}$	$\overline{\mathrm{IORQ}}$	$\overline{\mathrm{MRD}}$	$\overline{\mathrm{MWR}}$	$\overline{\mathrm{IORD}}$	$\overline{\mathrm{IOWR}}$
L	H	L	H	L	H	H	H
H	L	L	H	H	L	H	H
L	H	H	L	H	H	L	H
H	L	H	L	H	H	H	L

このようにメモリとI/Oがそれぞれ専用のリード/ライト信号をもつことができる

〈図3-9〉 メモリ，I/O専用リード/ライト信号の合成法

これにより，アドレス・デコーダの入力をアドレス信号だけにできます．アドレス信号はほかのコントロール信号に先立って出力されるため，アクセス速度で余裕をもつことができます．詳細については，個々の素子との具体的な接続例の説明で行います．

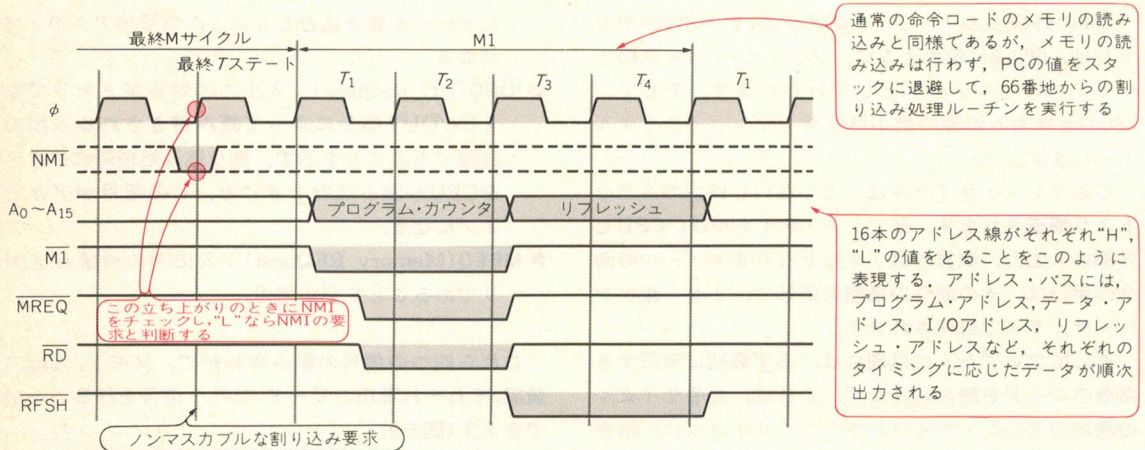
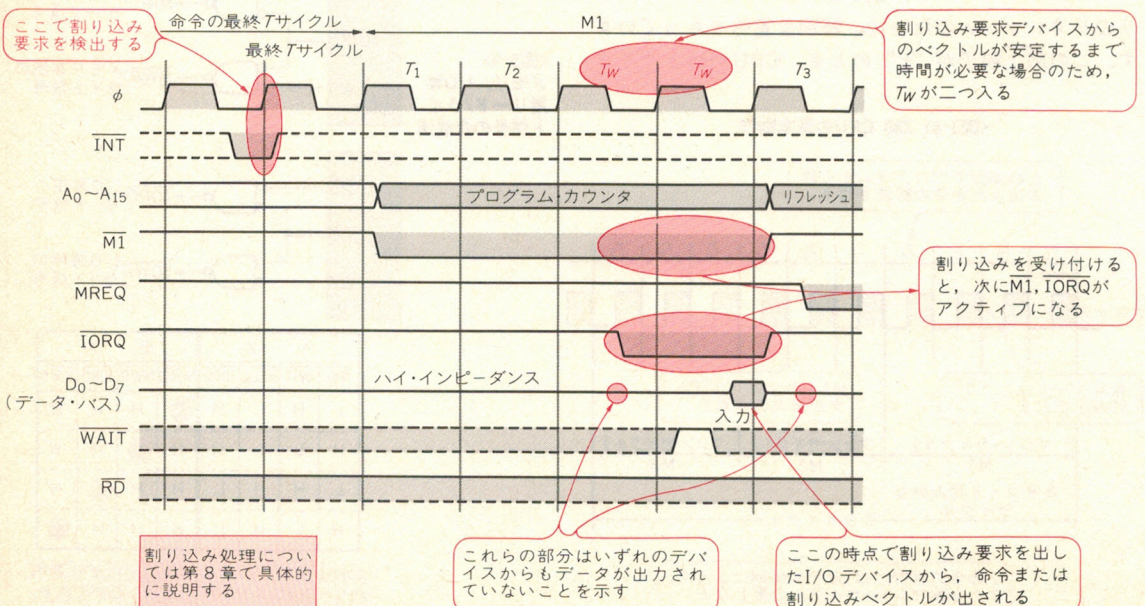
● 割り込み動作時に必要となる信号端子に関する説明

Z80の割り込みのための信号端子として，二種類の割り込み入力端子をもっています．

▶$\overline{\mathrm{NMI}}$(Non Maskable Interrupt)：ソフトウェアで割り込みの受け付けを禁止することのできない割り込み用の端子．電源異常などのシステムとしての動作を継続するのに重大な支障が生じたことをCPUに知らせる場合などに使用される．この機能を利用しないときは，この端子を抵抗を介して5Vにプルアップしておく(図3-10)．

▶$\overline{\mathrm{INT}}$(INTerrupt)：通常の割り込みを受け付けるための端子．この端子に接続される割り込み要求はソフトウェアなどによって，割り込みの受け付けを禁止することができる．システムは割り込みの処理ができるようになっていても，特別に連続処理が必要で，割り込み処理によって実行を中断されては困る場合に便利な機能である(図3-11)．

割り込みの種類，またその扱いが面倒な面が多々あります．具体的な例も含めて詳細は，第8章で説明し

〈図3-10〉ノン・マスカブル割り込み，要求動作

〈図3-11〉割り込み要求/アクノリッジ・サイクル

ます．

● DMA処理に関する信号端子についての説明

Z80 CPUは，CPU以外の装置または，素子が直接メモリなどにアクセスすることのできる，DMA(ダイレクト・メモリ・アクセス)のモードをもっています．この処理のために，$\overline{\mathrm{BUSRQ}}$(BUS ReQuest)，$\overline{\mathrm{BUSAK}}$(BUS AcKnowlege)の二つの信号端子があります．

▶$\overline{\mathrm{BUSRQ}}$：外部からCPUにDMAを要求するための信号端子．この信号がアクティブになると，現在，実行中のマシン・サイクルが終わりしだい，DMAモードになる．データ・バス，アドレス・バス，コントロール・バスのうち，入出力に関係する$\overline{\mathrm{RD}}$，$\overline{\mathrm{WR}}$，$\overline{\mathrm{MREQ}}$，$\overline{\mathrm{IORQ}}$のバスがハイ・インピーダンスの状態になる．

▶$\overline{\mathrm{BUSAK}}$：外部からのDMAの要求に対する応答で，DMAのモードになったことを示すための出力信号．CPUチップからの出力をバッファで増強している場合，この信号でコントロール・バスのバッファも含めてハイ・インピーダンスにする．

● 外部素子との同期のための信号

▶$\overline{\mathrm{WAIT}}$：このCPUチップに接続される素子は，必ずしもCPUの動作と同じではない．CPUの処理速度は，より高速な処理の要求から，高い周波数のシステム・クロックを使用する場合が多くなっている．

この場合，外部の素子の処理速度が追随できないことが生じる．これに対してCPUは，外部のROM，I/O装置の処理の遅れを待つことができる(図3-12)．CPUは，T_2ステートのクロックの立ち下がりで$\overline{\mathrm{WAIT}}$端子の状態を検出し，アクティブなら次のステートをT_W(ウェイト・ステート)として，$\overline{\mathrm{WAIT}}$端子をアクティブにした素子の処理が進み，$\overline{\mathrm{WAIT}}$端子をインアクティブにするまでウェイト・ステートの挿入が続く．

● ダイナミック・メモリのリフレッシュのための信号

▶$\overline{\mathrm{RFSH}}$(ReFreSH)：ダイナミック・メモリのリフレッシュのためのアドレス信号をアドレス・バスに出力していることを示す．

Z80は，この$\overline{\mathrm{RFSH}}$の出力されている期間に，アドレスのA_0〜A_6の7ビットのリフレッシュのためのアドレス信号が出力されます．一般的な64Kビットまでのダイナミック・メモリは，アドレス・ラインのA_0〜A_6の7ビットのアドレスで示される128列のメモリ・セルを順番にアクセスすることでリフレッシュが行われます．2 ms以内にA_0〜A_6で示されるすべての

これだけは　知っておきたい

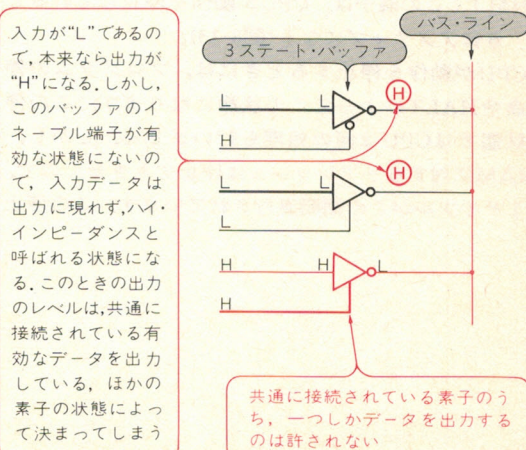
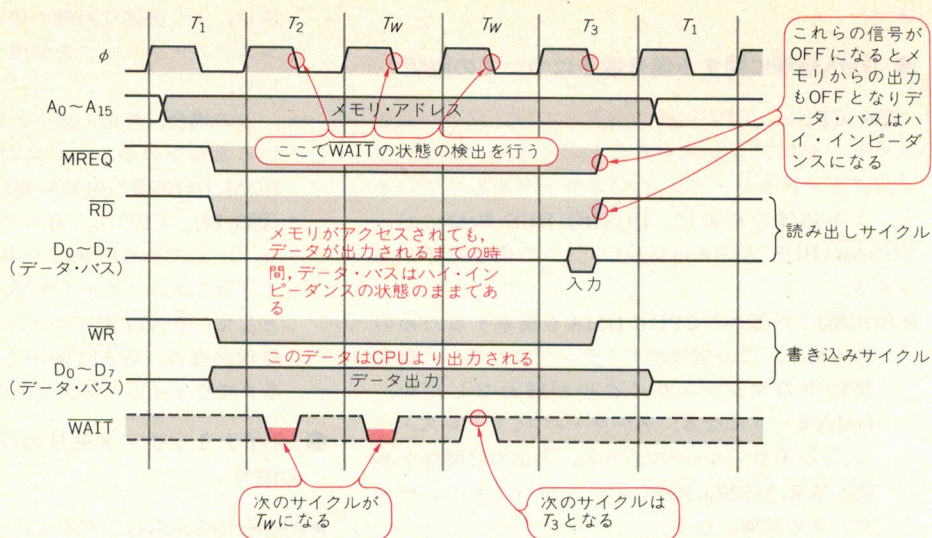
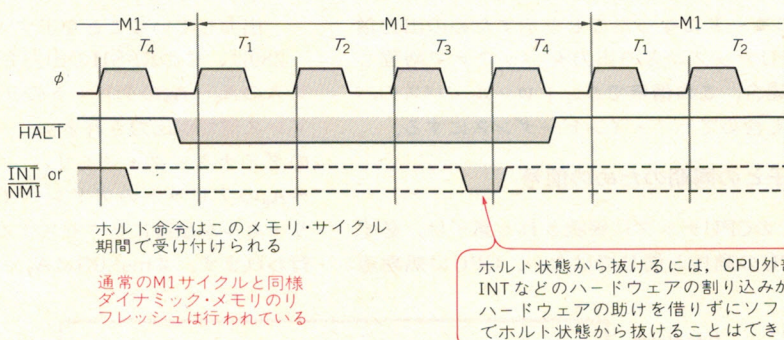
3ステート・バッファ

ディジタル回路では，一般に“0”，“1”または，“H”，“L”の二つの状態しか意味をもっていません．したがって，ディジタル回路で使用する各素子の出力は，“H”または“L”の有意な値を出力します．しかし，コンピュータ・システムの各バスに接続する素子には，“H”，“L”以外のもう一つの出力状態をもった3ステートと呼ばれるものがあります．

これは，コンピュータ・システムのデータ・バスのように，同じ信号ラインに複数の出力素子が接続されている場合，その複数の素子が同時に出力状態にあると，正しい出力状態を示すことができません．そのとき，出力すべき素子以外は“H”，“L”のいずれの状態も出力しないハイ・インピーダンスの状態になります．

したがって，共通のバスに多く素子が接続されていても，そのとき選択された素子のみがデータを出力することができます．$\overline{\mathrm{CE}}$または$\overline{\mathrm{OE}}$などの信号は，この出力バッファのコントロールを行っています．

メモリ，マイクロコンピュータ用の各種の周辺デバイスの出力は，この3ステートの出力になっているのが普通です．

〈図3-B〉 3ステート出力を共通線につないだときの状態

〈図3-12〉
待ち状態を含むメモリの読み出しと書き込みのサイクル

〈図3-13〉
ホルト状態解除

アドレスについてリフレッシュを行います．

この機能があるため，Z80はダイナミック・メモリを用いた大容量メモリのシステムが容易に実現できます．

● CPUの状態表示

▶$\overline{\text{HALT}}$：この端子は，CPUが動作を停止しているときにアクティブになる(図3-13)．

CPUが動作を停止するときには，プログラムの停止命令HALTによってこの状態になります．この停止状態ではCPUは何の処理も行いません．しかし，DRAMに対するリフレッシュは停止できませんので，M1サイクルがこの期間実行されています．この停止状態から抜け出すには，次の二つの方法があります．

(1) リセットにより，0番地からのプログラムを実行する．

(2) 割り込みにより，それぞれの割り込み処理ルーチンへ抜ける．

何れの方法を用いても，外部よりハードウェア上の処理が加わる必要があります．CPUが動作を停止しているため，次の命令を読み込むこともできません．したがって，このHALT状態から抜け出す命令はありません．

以上説明したようにZ80は，多くの機能をもっています．以後各章でそれぞれの機能の詳細について具体的に説明していきます．

メモリとの接続

■ NEXT

CPUと各種メモリとの接続方法を示します．Z80システムに使われるメモリの特徴，アドレス・デコードの具体的方法などについても，詳しく説明します．

keywords

DRAM　：ダイナミックRAM．トランジスター一つでセルを構成する高集積度のメモリ．半導体摩擦のきっかけとなったメモリ．

リフレッシュ：DRAMの内容が消えないように，通常2msごとに繰り返されるアクセス．

アクセス・タイム：メモリの読み出しを開始してから，データが出力端子に表れるまでの時間．

マスクROM　：製造時に内容も書き込まれるROM．量産時には安価になる．

P-ROM　：プログラマブルROM．使用時に任意の内容が書き込めるROM．

UVEP-ROM　：紫外線を照射することで内容を消去し，繰り返し使用可能なROM．

CE　：chip enable．並列にバスに接続されたデバイスを選択するための端子．

OE　：output enable．3ステートの出力バッファを出力にするための端子．

SRAM：書き込んだら電源を切らない限り内容が変わらないメモリ．

フロート：3ステートのバッファで，出力が電気的に切り離されて，状態が定まっていない状態．

コンピュータのシステムでは，メモリは不可欠な存在です．このメモリは，現在，半導体メモリが使用されています．そしてこの半導体メモリは，最も技術の進展の激しい素子です．このメモリの集積度の向上がビット当たりの単価を下げ，安価なマイクロコンピュータ・システムでも容易に大掛かりなソフトウェアを利用できるようになりました．その結果ますます多くの分野で，これらマイクロコンピュータ・システムの利用が拡大しています．

本章では，これらメモリの中でとくに8ビットCPUでよく利用されるものについて，接続法などについて具体的に説明します．

メモリの種類

メモリには，そのデータの記憶形態によって，ROM, RAMの二種類に大別できます．また，一度にアクセスできるデータの幅，ビット数もメモリの種類によって異なっています．

ROMは，主にプログラムを格納するのに利用されます．一度に読み書きできるデータの幅は，8ビットCPUと同じ8ビットのものが通常使用されます．試作などには，27×××と呼ばれる，紫外線照射によってプログラムの消去可能なUVEP-ROMがよく利用されます(図4-1)．量産する場合は，このROMと差し換えができるコンパチブルなマスクROMを利用することができます．

RAMは，前述のROMとピン・コンパチブルなスタティックRAMが小容量のメモリ・システムでは使用されます．大容量のRAMを必要とする場合は，ダイナミック・メモリを使用することになります．

ダイナミック・メモリの多くは，それぞれのチップが1ビットのデータ幅しかありません．そのため，バイト単位のデータを取り扱うには，8個を並列に接続する必要があります．Z80 CPUの場合は，64KビットDRAMを8個使用して，64Kバイトのメモリを容易に作成できます．

現在では256KビットDRAMも多く使用されています．しかし，通常のアプリケーションでしたら，64Kバイトまででほとんど間に合います．また，Z80の8ビットCPUのアドレス空間は，64Kバイトまでですので，それ以上のメモリ空間が必要となる場合は，バンク・セレクトなどの新たな技術が必要になります．

図4-2, 図4-3にそれぞれCPUとの接続法の具体的な例を示しておきます．

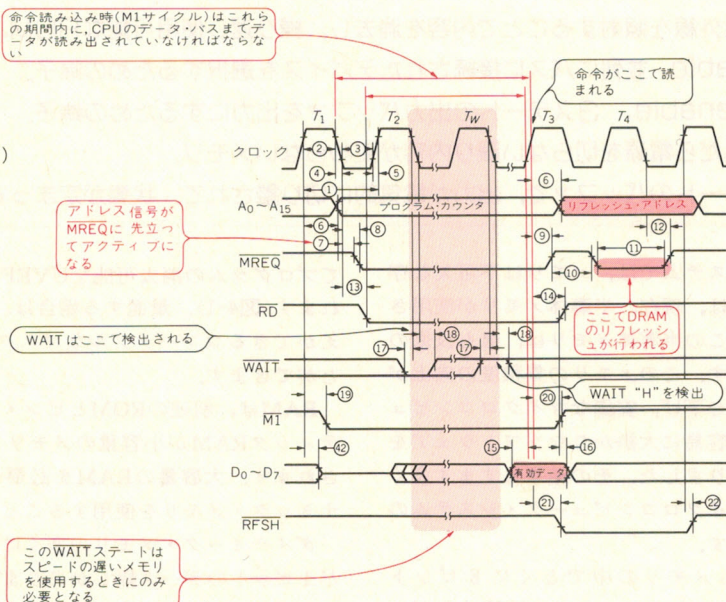
〈図4-1〉
8ビットCPUによく使用されるROM，RAMのピン配置

512K	256K	128K	64K	64K	32K	16K	16K	容量(ビット)
27512	27256	27128	2764	TC5564	2732	2716	6116 5516	ピン
A_{15}	V_{PP}	V_{PP}	V_{PP}	NC				1
A_{12}	A_{12}	A_{12}	A_{12}	A_{12}				2
A_7	A_7	A_7	A_7	A_7	A_7	A_7	A_7	3(1)
A_6	A_6	A_6	A_6	A_6	A_6	A_6	A_6	4(2)
A_5	A_5	A_5	A_5	A_5	A_5	A_5	A_5	5(3)
A_4	A_4	A_4	A_4	A_4	A_4	A_4	A_4	6(4)
A_3	A_3	A_3	A_3	A_3	A_3	A_3	A_3	7(5)
A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	8(6)
A_1	A_1	A_1	A_1	A_1	A_1	A_1	A_1	9(7)
A_0	A_0	A_0	A_0	A_0	A_0	A_0	A_0	10(8)
D_0	D_0	D_0	D_0	D_0	D_0	D_0	D_0	11(9)
D_1	D_1	D_1	D_1	D_1	D_1	D_1	D_1	12(10)
D_2	D_2	D_2	D_2	D_2	D_2	D_2	D_2	13(11)
GND	GND	GND	GND	GND	GND	GND	GND	14(12)

ピン	6116 5516	2716	2732	TC5564	2764	27128	27256	27512
容量(ビット)	16K	16K	32K	64K	64K	128K	256K	512K
28				V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}
27				$R/\overline{W}$	$\overline{PGM}$	$\overline{PGM}$	A_{14}	A_{14}
(24)26	V_{CC}, V_{DD}	V_{CC}, V_{DD}	V_{CC}, V_{DD}	CE_2	NC	A_{13}	A_{13}	A_{13}
(23)25	A_8	A_8	A_8	A_8	A_8	A_8	A_8	A_8
(22)24	A_9	A_9	A_9	A_9	A_9	A_9	A_9	A_9
(21)23	$R/\overline{W}$	V_{PP}	A_{11}	A_{11}	A_{11}	A_{11}	A_{11}	A_{11}
(20)22	$\overline{CE_1}$*	$\overline{CS}$	$\overline{OE}/V_{PP}$	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$	$V_{PP}/\overline{OE}$
(19)21	A_{10}	A_{10}	A_{10}	A_{10}	A_{10}	A_{10}	A_{10}	A_{10}
(18)20	$\overline{CE_2}$*	PD/PGM	$\overline{CE}$	$\overline{CE_1}$	$\overline{CE}$	$\overline{CE}$	$\overline{CE}$	$\overline{CE}$
(17)19	D_7	D_7	D_7	D_7	D_7	D_7	D_7	D_7
(16)18	D_6	D_6	D_6	D_6	D_6	D_6	D_6	D_6
(15)17	D_5	D_5	D_5	D_5	D_5	D_5	D_5	D_5
(14)16	D_4	D_4	D_4	D_4	D_4	D_4	D_4	D_4
(13)15	D_3	D_3	D_3	D_3	D_3	D_3	D_3	D_3

＊6116は，⑳が$\overline{OE}$，⑱が$\overline{CE}$

(上から見た図)

〈図4-4〉
Z80のメモリ・リード
(オペコード・フェッチ)

タイミング

メモリを接続して正常に動作させるためには，配線の接続だけでなく，具体的な動作時のタイミングについても考慮しなければなりません．

● メモリのアクセス・タイム

メモリ中のデータを読み出そうとするとき，**メモリの読み出し作業を開始してからデータの出力端子にそのメモリ中のデータが現れるまでに，必ず所定の時間が必要**となります．また，メモリのコントロール端子を決まった手順に従って，制御しなければなりません．

これも，RAMとROMでは少し異なっています．またDRAMの場合は，DRAMの内部処理の関係から，多くのことを考慮しなければなりません．しかし，基本的な事項としてはいずれにも共通して，次のアクセス・タイムについて検討する必要があります．

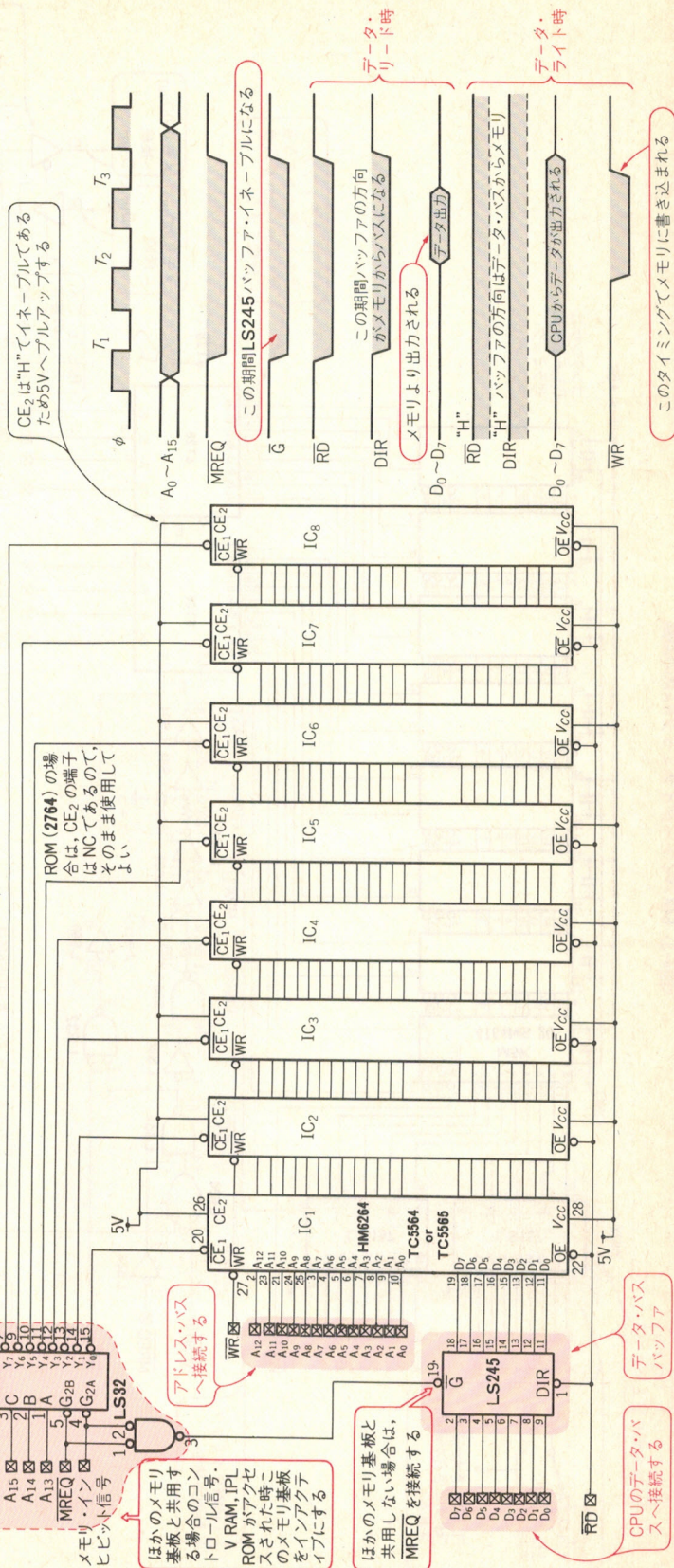
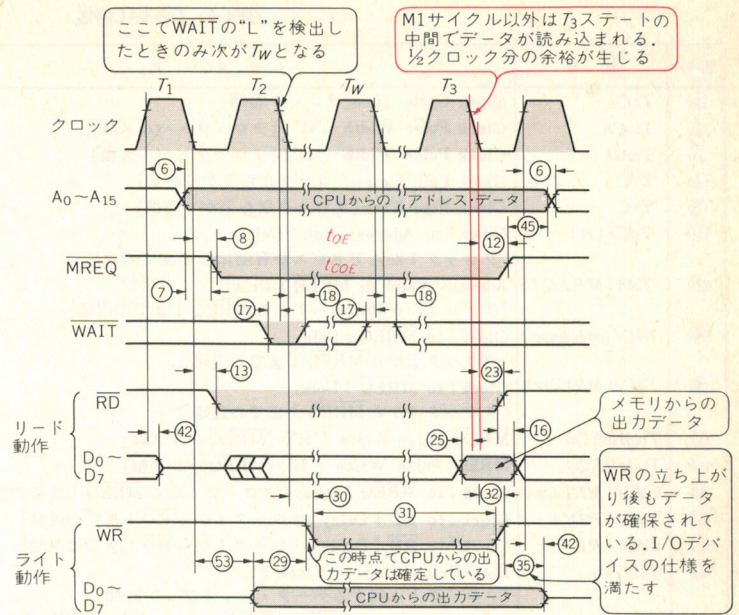
データの読み込み開始から，出力端子へデータが現れ確定するまで，メモリによって特定の時間が必要であるということから，CPUの動作スピード，メモリのアクセス・タイムの関連を検討しないと，正しい動作が保証されないことがあります(**図4-4，図4-5，図4-6，表4-1，表4-2**)．

〈図4-2(a)〉 64Kビット・スタティック RAMを使用したメモリ基板

$\overline{MREQ}$	メモリ・インヒビット	A_{15}	A_{14}	A_{13}	選択されるIC	各メモリのアドレス
L	L	L	L	L	IC_1	0000H～1FFFH
L	L	L	L	H	IC_2	2000H～3FFFH
L	L	L	H	L	IC_3	4000H～5FFFH
L	L	L	H	H	IC_4	6000H～7FFFH
L	L	H	L	L	IC_5	8000H～9FFFH
L	L	H	L	H	IC_6	A000H～BFFFH
L	L	H	H	L	IC_7	C000H～DFFFH
L	L	H	H	H	IC_8	E000H～FFFFH
H	×	×	×	×	このメモリはアクセスされない	
×	H	×	×	×	このメモリはアクセスされない	

×はH,Lいずれでもよい

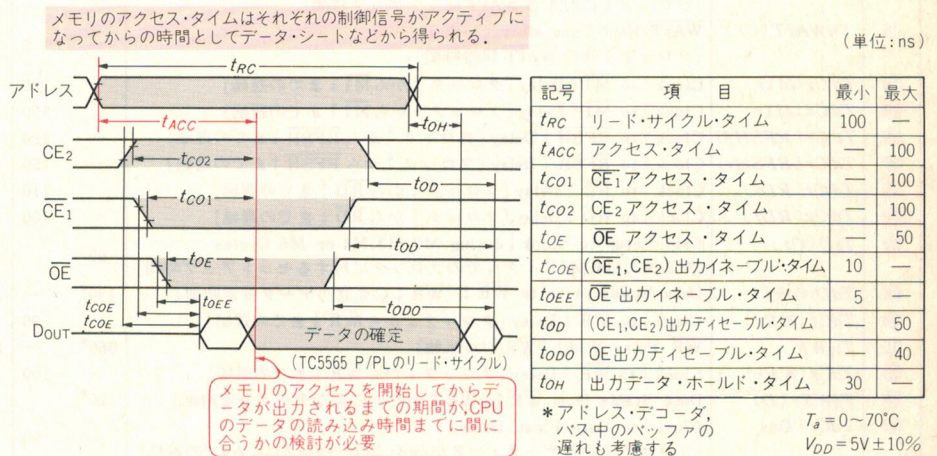
〈図4-2(b)〉 バッファの制御タイミング

〈図4-3〉64Kバイト・ダイナミックRAMの回路

〈図4-5〉 メモリへのデータの読み書き

〈図4-6〉
メモリ側のタイミング
(スタティックRAM)

(単位:ns)

記号	項　目	最小	最大
t_{RC}	リード・サイクル・タイム	100	—
t_{ACC}	アクセス・タイム	—	100
t_{CO1}	$\overline{CE_1}$ アクセス・タイム	—	100
t_{CO2}	CE_2 アクセス・タイム	—	100
t_{OE}	$\overline{OE}$ アクセス・タイム	—	50
t_{COE}	($\overline{CE_1}$, CE_2) 出力イネーブル・タイム	10	—
t_{OEE}	$\overline{OE}$ 出力イネーブル・タイム	5	—
t_{OD}	(CE_1, CE_2) 出力ディセーブル・タイム	—	50
t_{ODO}	OE出力ディセーブル・タイム	—	40
t_{OH}	出力データ・ホールド・タイム	30	—

*アドレス・デコーダ,バス中のバッファの遅れも考慮する

$T_a = 0 \sim 70°C$
$V_{DD} = 5V \pm 10\%$

CPUがアドレスを出力して,メモリからデータを読み込むタイミングまでに,データが確定していなければなりません.CPUの動作速度を上げていきますと,メモリがその処理速度に追いつかなくなってきます.その場合,CPUの動作をメモリなど動作速度の遅いデバイスに合わせるために,待たせることができるようになっています.これはZ80ではT_2ステートの後にT_Wというウェイト(待ち)ステートをCPUの動作に挿入することで実現できます.

● P-ROMの場合のタイミング

図4-7,表4-3に,主なP-ROMのタイミングを示しておきます.

● 各メモリのタイミング

メモリのタイミングについて検討すべき基本的な項目は,次のようなことです.

CPUからメモリに対して与えた,アドレス,$\overline{CE}$,$\overline{OE}$などのコントロール信号が,メモリの各端子に到達します.その時点でメモリがアクティブになり,データが出力されます.そして,そのデータがデータ・バス上のバッファを経由して,CPUのデータ・バスの端子に到達します.

このアドレス出力から,データ到達までの時間が,CPUのデータ取り込みのタイミングまでに,間に合うかどうかの問題となります.

以前は,ROMなどのスピードも遅かったので,ウェイト・ステートの挿入などの工夫を必要としました.しかし,最近はROMでもかなり高速ですので,Z80の4MHzバージョンでもウェイトなしで動作させる

〈表4-1〉 Z80のAC特性

番号	記号	パラメータ	Z80 CPU min	Z80 CPU max	Z80A CPU min	Z80A CPU max	Z80B CPU min	Z80B CPU max
①	*TcC*	Clock Cycle Time〔クロック周期〕	400*		250*		165*	
②	*TwCh*	Clock Pulse Width ("H")〔クロック・パルス幅〕	180*		110*		65*	
③	*TwCl*	Clock Pulse Width ("L")〔クロック・パルス幅〕	180	2000	110	2000	65	2000
④	*TfC*	Clock Fall Time〔クロック立ち下がり時間〕	—	30	—	30	—	20
⑤	*TrC*	Clock Rise Time〔クロック立ち上がり時間〕		30		30		20
⑥	*TdCr(A)*	Clock↑to Address Valid Delay 〔クロック↑からアドレスが有効になるまでの遅延〕	—	145	—	110	—	90
⑦	*TdA(MREQf)*	Address Valid to $\overline{MREQ}$↓Delay 〔アドレスが有効になってから$\overline{MREQ}$↓までの遅延〕	125*	—	65*	—	35*	—
⑧	*TdCf(MREQf)*	Clock↓to $\overline{MREQ}$↓Delay 〔クロック↓から$\overline{MREQ}$↓までの遅延〕	—	100	—	85	—	70
⑨	*TdCr(MREQr)*	Clock↑to $\overline{MREQ}$↑Delay 〔クロック↑から$\overline{MREQ}$↑までの遅延〕	—	100	—	85	—	70
⑩	*TwMREQh*	$\overline{MREQ}$ Pulse Width ("H")〔$\overline{MREQ}$パルス幅〕	170*		110*		65*	
⑪	*TwMREQl*	$\overline{MREQ}$ Pulse Width ("L")〔$\overline{MREQ}$パルス幅〕	360*	—	220*	—	135*	—
⑫	*TdCf(MREQr)*	Clock↓to $\overline{MREQ}$↑Delay〔クロック↓から$\overline{MREQ}$↑までの遅延〕	—	100	—	85	—	70
⑬	*TdCf(RDf)*	Clock↓to $\overline{RD}$↓Delay〔クロック↓から$\overline{RD}$↓までの遅延〕	—	130	—	95	—	80
⑭	*TdCr(RDr)*	Clock↑to $\overline{RD}$↑Delay〔クロック↑から$\overline{RD}$↑までの遅延〕	—	100	—	85	—	70
⑮	*TsD(Cr)*	Data Setup Time to Clock↑ 〔クロック↑に対するデータ・セットアップ時間〕	50		35		30	
⑯	*ThD(RDr)*	Data Hold Time to $\overline{RD}$↑〔$\overline{RD}$↑に対するデータ保持時間〕	—	0	—	0	—	0
⑰	*TsWAIT(Cf)*	$\overline{WAIT}$ Setup Time to Clock↓ 〔クロック↓に対する$\overline{WAIT}$セットアップ時間〕	70	—	70	—	60	—
⑱	*ThWAIT(Cf)*	$\overline{WAIT}$ Hold Time after Clock↓ 〔クロック↓後の$\overline{WAIT}$保持時間〕	—	0	—	0	—	0
⑲	*TdCr(M1f)*	Clock↑to $\overline{M1}$↓Delay〔クロック↑から$\overline{M1}$↓までの遅延〕	—	130	—	100	—	80
⑳	*TdCr(M1r)*	Clock↑to $\overline{M1}$↑Delay〔クロック↑から$\overline{M1}$↑までの遅延〕		130		100		80
㉑	*TdCr(RFSHf)*	Clock↑to $\overline{RFSH}$↓Delay〔クロック↑から$\overline{RFSH}$↓までの遅延〕	—	180	—	130	—	110
㉒	*TdCr(RFSHr)*	Clock↑to $\overline{RFSH}$↑Delay〔クロック↑から$\overline{RFSH}$↑までの遅延〕	—	150	—	120	—	100
㉓	*TdCr(RDr)*	Clock↓to $\overline{RD}$↑Delay〔クロック↓から$\overline{RD}$↑までの遅延〕	—	110	—	85	—	70
㉔	*TdCr(RDf)*	Clock↑to $\overline{RD}$↓Delay〔クロック↑から$\overline{RD}$↓までの遅延〕	—	100	—	85	—	70
㉕	*TsD(Cf)*	Data Setup to Clock↓during M2, M3, M4 or M5 Cycles 〔M2, M3, M4, M5サイクルでのクロックに対するセットアップ時間〕	60		50		40	
㉙	*TdD(WRf)*	Data Stable prior to $\overline{WR}$↓〔$\overline{WR}$↓に先立つデータ安定時間〕	190*	—	80*	—	25*	—
㉚	*TdCf(WRf)*	Clock↓to $\overline{WR}$↓Delay〔クロック↓から$\overline{WR}$↓までの遅延〕		90		80		70
㉛	*TwWR*	$\overline{WR}$ Pulse Width〔$\overline{WR}$パルス幅〕	360*	—	220*	—	135*	—
㉜	*TdCf(WRr)*	Clock↓to $\overline{WR}$↑Delay〔クロック↓から$\overline{WR}$↑までの遅延〕	—	100	—	80	—	70
㉟	*TdWRr(D)*	Data Stable from $\overline{WR}$↑〔$\overline{WR}$↑からの必要なデータ安定時間〕	120*		60*		30*	
㊷	*TdCr(Dz)*	Clock↑to Data Float Delay 〔クロック↑からデータ・バスがハイ・インピーダンスになるまでの遅延〕	—	90	—	90	—	80
㊺	*TdCTr(A)*	$\overline{MREQ}$↑, $\overline{IORQ}$↑, $\overline{RD}$↑, and 〔$\overline{MREQ}$↑, $\overline{IORQ}$↑, $\overline{RD}$↑, $\overline{WR}$↑からのアドレス保持時間〕	160*		80*		35*	
㊿	*TdCf(D)*	Clock↓to Data Valid Delay 〔クロック↓からデータが有効になるまでの遅延〕	—	230	—	150	—	130

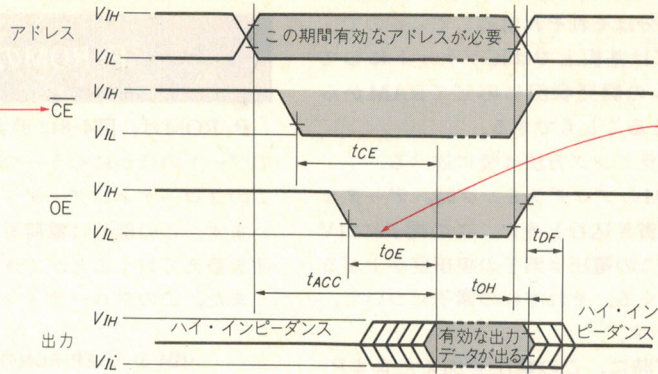
*クロック・サイクルの値に依存する. したがって個々の値は**表4-2**に従って計算する.

番号	記号	Z80	Z80A	Z80B
1	*TcC*	*TwCh*+*TwCl*+*TrC*+*TfC*	*TwCh*+*TwCl*+*TrC*+*TfC*	*TwCh*+*TwCl*+*TrC*+*TfC*
2	*TwCh*	*TwCh*は200μs以下でなければならない	*TwCh*は200μs以下でなければならない	*TwCh*は200μs以下でなければならない
7	*TdA(MREQf)*	*TwCh*+*TfC*−75	*TwCh*+*TfC*−65	*TwCh*+*TfC*−50
10	*TwMREQh*	*TwCh*+*TfC*−30	*TwCh*+*TfC*−20	*TwCh*+*TfC*−20
11	*TwMREQl*	*TcC*−40	*TcC*−30	*TcC*−30
26	*TdA(IORQf)*	*TcC*−80	*TcC*−70	*TcC*−55
29	*TdD(WRf)*	*TcC*−210	*TcC*−170	*TcC*−140
31	*TwWR*	*TcC*−40	*TcC*−30	*TcC*−30
33	*TdD(WRf)*	*TwCl*+*TrC*−180	*TwCl*+*TrC*−0 140	*TwCl*+*TrC*−140
35	*TdWRr(D)*	*TwCl*+*TrC*−80	*TwCl*+*TrC*−70	*TwCl*+*TrC*−55
45	*TdCTr(A)*	*TwCl*+*TrC*−40	*TwCl*+*TrC*−50	*TwCl*+*TrC*−50
50	*TdM1f(IORQf)*	2*TcC*+*TwCh*+*TfC*−80	2*TcC*+*TwCh*+*TfC*−65	2*TcC*+*TwCh*+*TfC*−50

〈表4-2〉
AC特性に対する補足説明

ACテスト条件

$V_{IH}=2.0V$
$V_{IL}=0.8V$
$V_{IHC}=V_{CC}\sim0.6V$
$V_{ILC}=0.45V$
$V_{OH}=2.0V$
$V_{OL}=0.8V$
$FLOAT=\pm0.5V$

〈図4-7〉 P-ROMのリード・タイミング

〈表4-3〉
図4-7に示すインテル社の各P-ROMのアクセス・スピード

パラメータ ＼ 各素子のバージョン				2764-2		2764-25 2764		2764-30 2764-3		2764-45 2764-4		単位	テスト条件
		2764A-1				2764A-25 2764A		2764A-30 2764A-3		2764A-45 2764A-4			
						27128-25 27128		27128-30 27128-3		27128-45 27128-4			
						27256-25 27256		27256-30 27256-3		27256-45 27256-4			
		最小	最大	最小	最大	最小	最大	最小	最大	最小	最大		
t_{ACC}	アドレスから出力までの遅延		180		200		250		300		450	ns	$\overline{\text{CE}}=\overline{\text{OE}}=V_{IL}$
t_{CE}	$\overline{\text{CE}}$から出力までの遅延		180		200		250		300		450	ns	$\overline{\text{OE}}=V_{IL}$
t_{OE}	$\overline{\text{OE}}$から出力までの遅延		65		75		100		120		150	ns	$\overline{\text{CE}}=V_{IL}$
t_{DF}	$\overline{\text{OE}}$が“H”になってから出力がフロートになるまで	0	55	0	60	0	85*	0	105	0	130	ns	$\overline{\text{CE}}=V_{IL}$
t_{OH}	アドレス,$\overline{\text{CE}}$,$\overline{\text{OE}}$のうちいずれかが最初にインアクティブになってからの出力の持続時間	0		0		0		0		0		ns	$\overline{\text{CE}}=\overline{\text{OE}}=V_{IL}$

＊各素子により若干異なる．64A；55, 128；60, 256；60

ことが可能です．

具体的なROM，スタティックRAMのリード/ライトのタイミング図をもとにして，CPUのクロックと比較し，タイミングの検討を行ってください．データ・シートなどにあるタイミングは，それぞれのメモリ・チップの入力端子と出力端子間についてのみ示してあります．

実際の回路では，アドレス・バスのバッファ，デコーダ，データ・バス上のバッファの信号の遅延が加わりますので，それらについても考慮しなければなりません．これらについては図の中に詳しく説明してあります．

制御端子の機能

ROMの場合は，次のような制御端子があります．

▶$\overline{\text{CE}}$：$\overline{\text{CE}}$(チップ・イネーブル)端子にアクティブな信号(この場合負論理なので“L”)が加えられると，このデバイスが活動(アクティブ)状態になる．ほかのコントロール端子にどのような信号が加わろうと，この端子がアクティブにならない限り，このデバイスは読み書きの動作を行わない．

素子によっては，バーのないCEで“H”のときに有意なイネーブル端子をもつものもある．

▶$\overline{\text{OE}}$：出力データ用のバッファをイネーブル(活動状態)にする端子．通常は，メモリ・セルから読み出されたデータがこの3ステートのバッファを通して出力される．このメモリが選択されていないときは，ほかの素子の出力と競合しないようにハイ・インピーダンスの状態にしておき，この素子が読み出しのために利用されるときにのみ，この端子をイネーブルにする．

前もってアドレス，$\overline{\text{CE}}$がイネーブルになっているなら，$\overline{\text{OE}}$からデータの出力までの時間はバッファをイネーブルにするだけなので，$\overline{\text{OE}}$から短いアクセス・タイムのみとなり，タイミングの設計が楽になる．ROMなどのとき，$\overline{\text{CE}}$はアドレス信号のみで作成すればよい．

▶$\overline{\text{PGM}}$：2764, 27128などのUVEP-ROMには，PGM

端子があり，2764とピン・コンパチブルなスタティックRAMでは，WRの書き込み信号用の端子となっている．この部分はそれぞれ対応しているので，使用方法によっては基板上ではICソケットにしておいて，システムの開発状況に応じてRAMからROMへと差し替えることもできる．

具体的なプログラミング方法は後に述べる．

▶ V_{PP}：UVEP-ROMのプログラミング時，データまたはプログラムを書き込むときに，この端子に21Vの電圧を加える．この電圧は素子の集積度が上がるにつれて変わってくる．それぞれの素子について，電圧値を表4-4に示す．

プログラミング時に，この電圧を加えたままP-ROMライタからICの抜き差しを行うと，メモリを壊すことがある．

● スタティックRAMの端子

$\overline{CE}$, $\overline{OE}$は，スタティックRAM, ROMともに同じ機能をもっています．スタティックRAMの場合はR/$\overline{W}$の端子ももっています．この端子は一般に，“H”のときにデータの読み出し，“L”のときにRAMへのデータの書き込みが行われます．

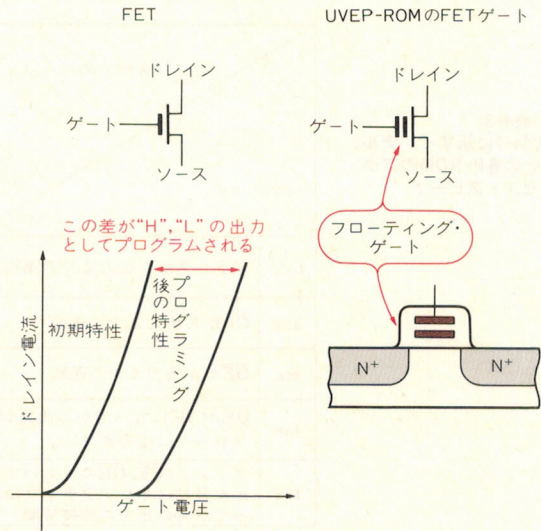
書き込みは，R/$\overline{W}$の立ち上がりの時点でのI/O端子上のデータが読み込まれます．TC5564(東芝)などの64Kビットのスタティック RAMの場合は，$\overline{CE}$と極性が反対のCE端子があります．

このCEのほうは一般的には5Vにプルアップしておきます．メモリのバッテリ・バックアップ時には，停電検出回路の出力をCEに接続します．このようにすれば，停電が検出されたらCEが“L”になり，メモリのアクセスを禁止してデータの保護ができます．

P-ROMの書き込み

P-ROMは，図4-8に示すように通常のMOS FETのゲートのほかにもう一つのゲートをもっています．このフローティング・ゲートは，外部から絶縁されています．この部分に電荷を注入すると，長期間その電荷を蓄えておくことができます．

また，このフローティング・ゲートに電荷が蓄積す

〈図4-8〉 UVEP-ROMのフローティング・ゲート

〈表4-4〉 インテル社の各P-ROMモード選択

端子 / モード	$\overline{CE}$ (20)				$\overline{OE}$ (22)				$\overline{PGM}$ (27)				A_9 (24)				V_{PP} (1)				V_{CC} (28)				出力 (11〜13,15〜19)
	2764	64A	128	256	2764	64A	128	256	64	64A	128		64	64A	128	256	64	64A	128	256	64	64A	128	256	共通
Read (読み込み)	V_{IL}	V_{IL}	V_{IL}	V_{IL}	V_{IL}	V_{IL}	V_{IL}	V_{IL}	V_{IH}	V_{IH}	V_{IH}		×	×	×	×	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	D_{OUT}
Output Disable (出力禁止)	V_{IL}	V_{IL}	V_{IL}	V_{IL}	V_{IH}	V_{IH}	V_{IH}	V_{IH}	V_{IH}	V_{IH}	V_{IH}		×	×	×	×	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	ハイ・インピーダンス
Standby (スタンバイ)	V_{IH}	V_{IH}	V_{IH}	V_{IH}	×	×	×	×	×	×	×		×	×	×	×	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{PP1}	V_{CC}	ハイ・インピーダンス
プログラム	V_{IL}		V_{IL}		V_{IH}		V_{IH}		V_{IL}		V_{IL}		×		×		V_{PP1}		V_{PP1}		V_{CC}		V_{CC}		D_{IN}
ベリファイ	V_{IL}	V_{IL}	V_{IL}	V_{IH}	V_{IL}	V_{IL}	V_{IL}	V_{IL}	V_{IH}	V_{IH}	V_{IH}		×	×	×	×	V_{PP1}	V_{PP2}	V_{PP1}	V_{PP2}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	D_{OUT}
プログラム・インヒビット	V_{IH}	V_{IH}	V_{IH}	V_{IH}	×	×	×	V_{IH}	×	×	×		×	×	×	×	V_{PP1}	V_{PP2}	V_{PP1}	V_{PP2}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	ハイ・インピーダンス
Inteligent Identifier (IDコード読み出し)	V_{IL}	V_{IL}	V_{IL}	V_{IL}	V_{IL}	V_{IL}	V_{IL}	V_{IL}	V_{IH}	V_{IH}	V_{IH}		V_H	V_H	V_H	V_H	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	コード
Inteligent Programming (高速書き込み)	V_{IL}	V_{IL}	V_{IL}	V_{IL}	V_{IH}	V_{IH}	V_{IH}	V_{IH}	V_{IL}	V_{IL}	V_{IL}		×	×	×	×	V_{PP1}	V_{PP2}	V_{PP1}	V_{PP2}	[V_{CC}]	[V_{CC}]	[V_{CC}]	[V_{CC}]	D_{IN}

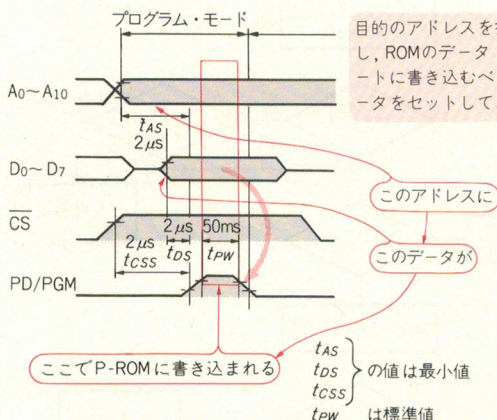
(注)×; V_{IL}, V_{IH}のどちらでもよい
V_H =12.0V±0.5V
V_{IL} =0.45V, V_{IH}=2.4V
V_{PP1}=21V±0.5V
V_{PP2}=12.5V±0.3V
V_{CC}=5.0V
[V_{CC}]=6.0V±0.25V

ることで，ソース-ドレイン間に必要な電流を流すためのゲート-ソース間電圧が，よけいに必要になります．このフローティング・ゲートでの電荷の蓄積によるゲート-ソース間電圧のしきい値の差によって，ビットのON/OFFをプログラミングします．

プログラミング時は，高電圧を印加することで，高エネルギ状態になった電子が絶縁を破ってフローティング・ゲートに蓄積することで行います．

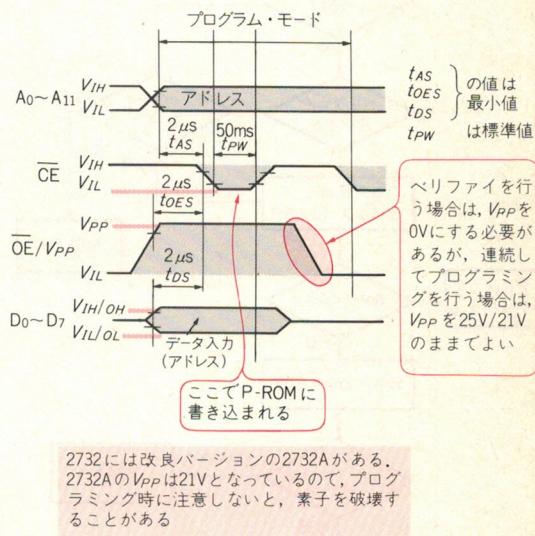
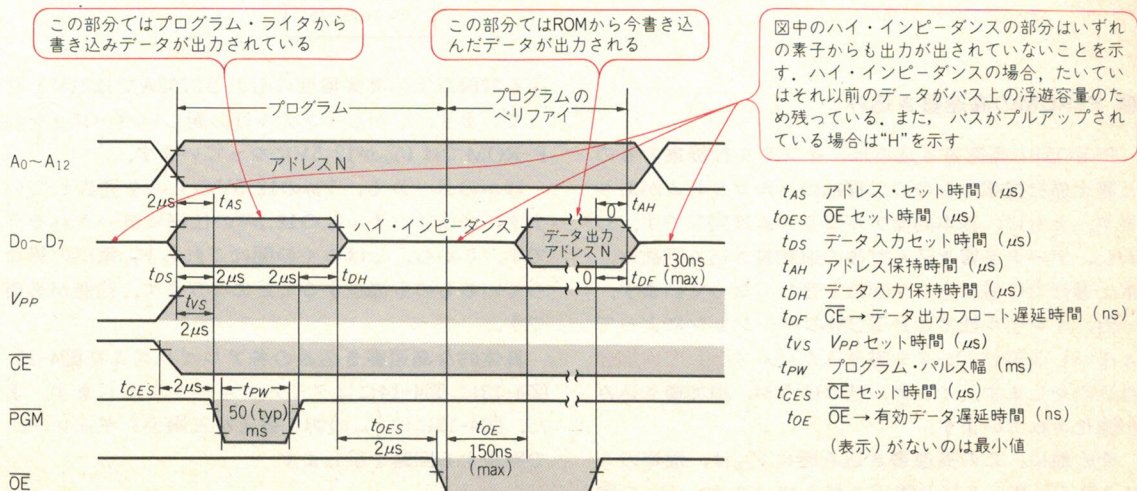
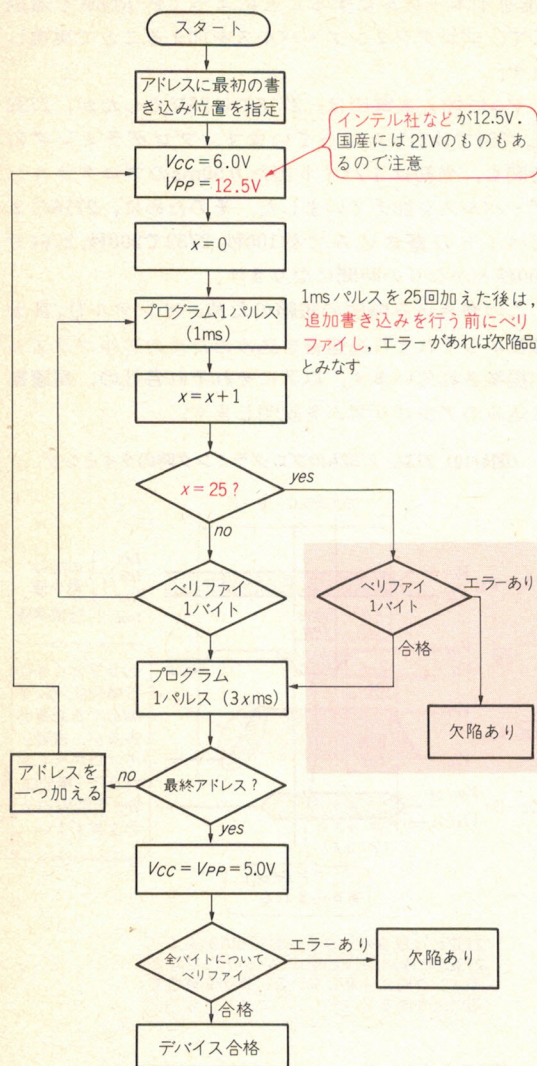
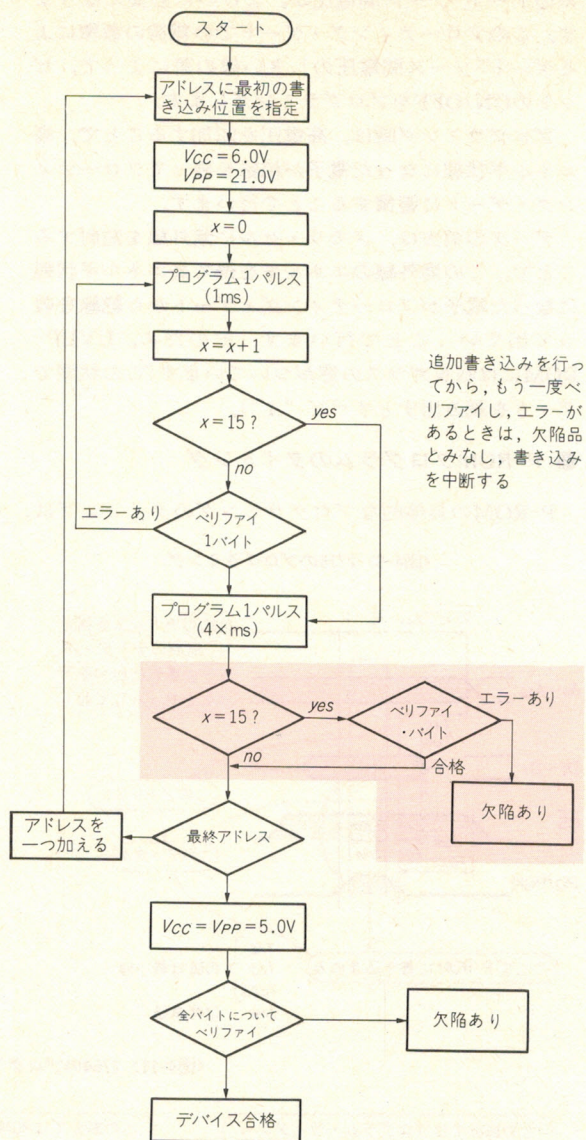
データの消去は，メモリ・セルに紫外線を照射することで，この紫外線のエネルギを得て高エネルギ状態になった電子がフローティング・ゲートから絶縁を破って出ていくことで行います．そのため，UVEP-ROMには石英ガラスの窓がついています(この状態でデータを読み出すとすべて“1”)．

● P-ROMプログラムのタイミング

P-ROMの具体的なプログラミングのタイミングは，図4-9～図4-11に示すようにV_{PP}端子に高電圧を加え，$\overline{CE}$をイネーブルにすることによってP-ROMを選択して，プログラミング・パルスを加えることで実現します．

V_{PP}に加える電圧は，2716では25Vでしたが，2732A，2764では21Vとなっています．プログラミングの時間も，当初は1バイト当たり50msのプログラミング・パルスを加えていました．そのために，2716の2Kバイトの書き込みで約100秒，2732で200秒，2764で400秒とかなりの時間になります．

この書き込み時間の短縮のため，インテル社，富士通社からそれぞれ高速書き込みのためのアルゴリズムが提案されています．以下にそれぞれ各社の，高速書き込みのアルゴリズムを説明します．

〈図4-9〉2716のプログラミング

〈図4-10〉2732，2732Aのプログラミング時のタイミング

〈図4-11〉2764のプログラミング時のタイミング

● P-ROMの高速書き込み

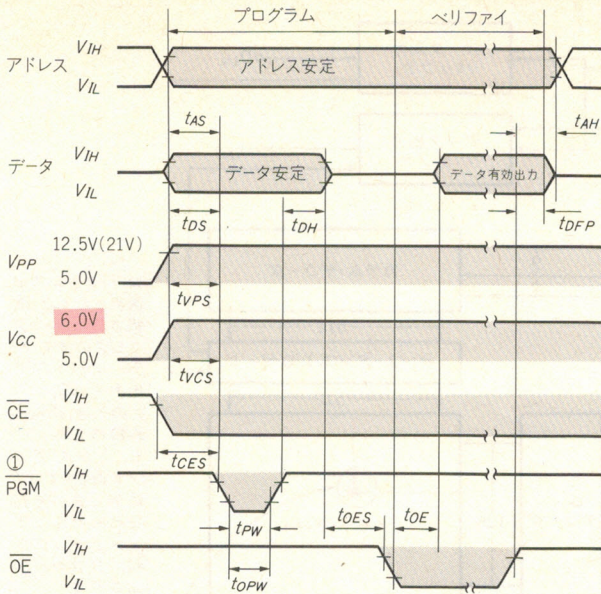
P-ROMの高速書き込みは，インテル社提案のものと富士通社提案のものの二種類のアルゴリズムがあります．ともに，基本的なアルゴリズムは同じです．しかし，データを書き込んだ後の追加書き込み回数が，富士通社では書き込んだ回数と等しくなっています．一方，インテル社ではP-ROMのバージョンによって4ないし3倍のパルスを書き込みパルスとして追加書き込みをします．インテル社のほうが，追加書き込みが強化されています．

その他に，この高速書き込み時にV_{CC}は，標準の5Vより1V高い6Vの電圧を加えています．V_{PP}の電圧も2764以上の高集積度のものと2732Aでは21Vとなっています．しかしインテル社の新しいバージョンのP-ROMではV_{PP}が12.5Vになっています．

ほかのメーカも，V_{PP}の12.5Vのものを発表しています．今後インテル社の12.5Vの仕様に統一されそうです．しかし，しばらくの間はこれらV_{PP}電圧の異なっているものも混在することになります．注意が必要です．

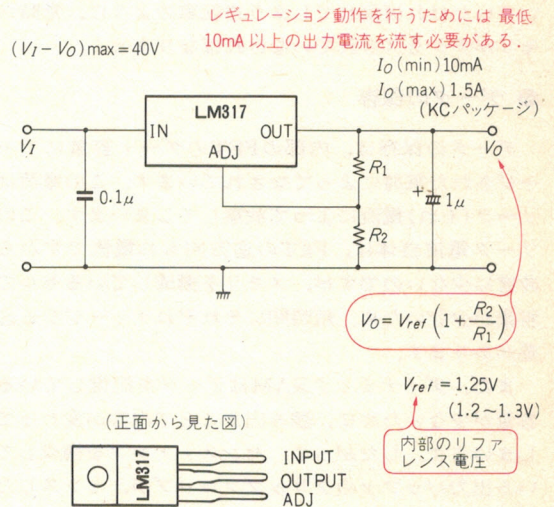
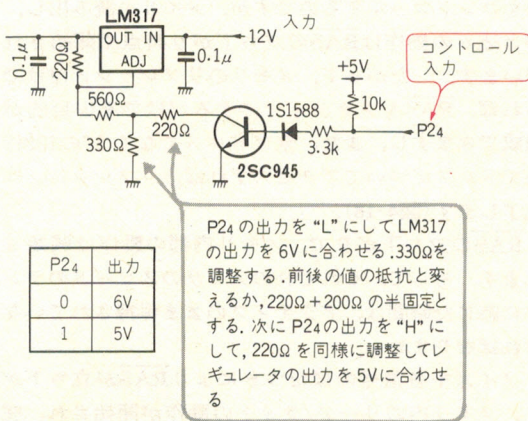
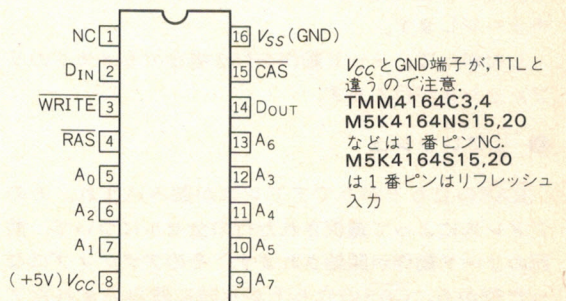
具体的な高速書き込みの各アルゴリズムを図4-12，図4-13に，図4-14にはタイムチャートを示します．また，図4-15にはV_{PP}の切り替えを三端子レギュレータで実現する回路を示します．

〈図4-14〉高速書き込みのタイミング

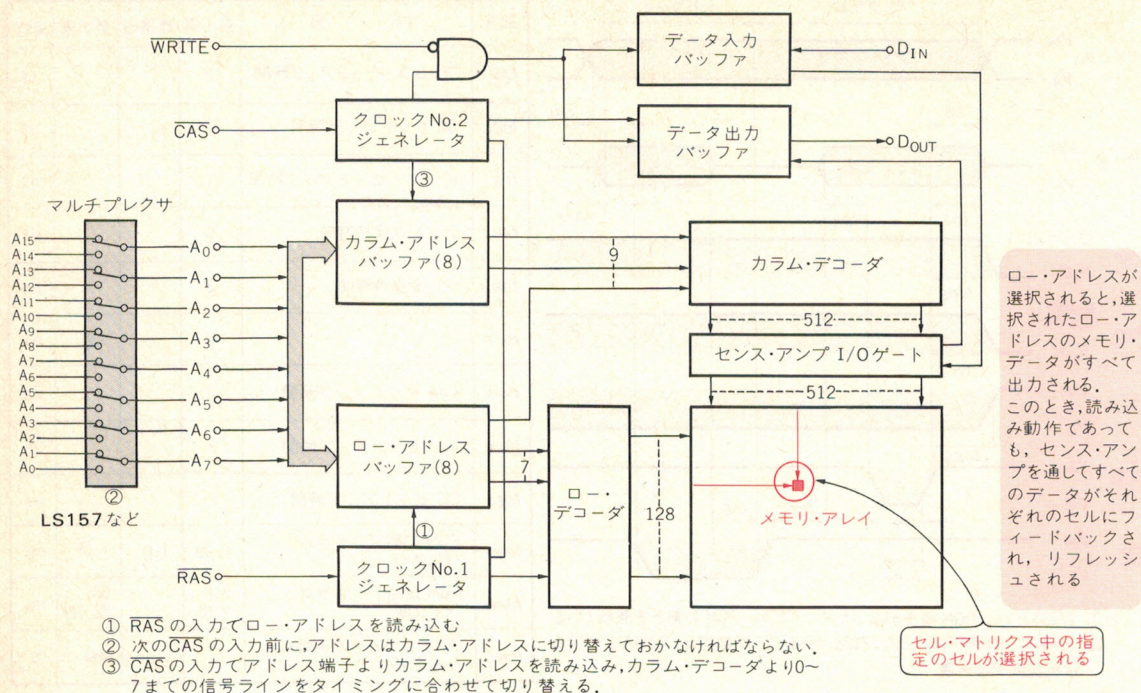
記号	内容	最小値	標準値	最大値	単位
t_{AS}	アドレス・セットアップ時間	2			μs
t_{OES}	$\overline{OE}$ セットアップ時間	2			μs
t_{DS}	データ・セットアップ時間	2			μs
t_{AH}	アドレス保持時間	0			μs
t_{DH}	データ保持時間	2			μs
t_{DFP}		0		130	ns
t_{VPS}	V_{PP} セットアップ時間	2			μs
t_{VCS}	V_{CC} セットアップ時間	2			μs
t_{CES}	$\overline{CE}$ セットアップ時間	2			μs
t_{PW}	$\overline{PGM}$ 初期プログラム・パルス幅	0.95	1.0	1.05	ms
t_{OPW}	$\overline{PGM}$ 追加プログラム・パルス幅	3.8 2.85		63 78.75	ms
t_{OW}	$\overline{OE}$ からデータが有効になるまで			150	ns

(注) ① 27256には，PGM端子がないため，$\overline{CE}$ が，$\overline{PGM}$ の働きを兼ねている
② t_{OPW} の上段数字は，2764, 27128, t_{OPW} の下段数字は，2764Aと27256，ほかの数字は共通

〈図4-15(a)〉三端子レギュレータLM317T

P2₄	出力
0	6V
1	5V

〈図4-15(b)〉V_{PP} の切り替え回路

〈図4-16〉64Kビット・ダイナミックRAMのピン配置図

ダイナミックRAM

大容量のメモリが必要な場合は，ダイナミック・メモリ(DRAM)を使用することになります(図4-16，図4-17)．現在DRAMとしては，64Kビットのものが一般的です．この64Kビットのものを8個使用すると，Z80 CPUの全メモリ・アドレス空間64Kバイトすべてをカバーできます．

〈図4-17〉ダイナミックRAMの内部構造(64Kビット)

ダイナミック・メモリは，名に示されるように，動いていなければ倒れてしまう自転車のように，常時メモリがアクセスされていなければなりません.

● データの保存

データの保存は，内部のFETのゲート容量にチャージされた電荷によってなされています．この電荷は，リーク(もれ)電流によって放電してしまいます．このリーク電流自体は，FETの逆方向もれ電流ですから非常に少ないのですが，メモリを構成しているセルの容量が少ないため，短時間にそれぞれチャージする必要があります.

また，ダイナミックRAMはデータを記憶している容量が少ないために，読み出すことで内容が変わってしまいます．したがって，センス・アンプを構成している出力バッファのフリップフロップが，センスした結果に基づいてその内容を確定し，外部にデータとして出力すると同時に，セルにも再び書き込み，セルをチャージします.

したがって，リード動作を行う場合でもメモリのリフレッシュが行えます.

● リフレッシュ

$\overline{RAS}$の立ち下がりでアドレスが読み込まれ，そのアドレスによって選択された行の全セルについて，前記のリード動作が開始されます．そのアクティブになった行から，$\overline{CAS}$の立ち下がり時に読み込まれたアドレスにしたがって，特定の列が選択されデータが出力されます.

$\overline{CAS}$は，列の選択を行うアドレスを読み込み，出力をコントロールするのですが，メモリの読み出し，書き込みの動作は$\overline{RAS}$の立ち下がり時点で開始されています．したがって，メモリのリフレッシュだけであれば，$\overline{RAS}$をアクティブにするだけでその目的が達成できますし，また，全アドレスでなく2^7の128行のアドレスについてアクセスすれば，リフレッシュは完了します(図4-18).

$\overline{RAS}$の立ち下がりで，メモリ内部の動作は開始されます．そして，それぞれのメモリのスピードのランクに応じた時間は，アクティブのまま保持されていなければなりません.

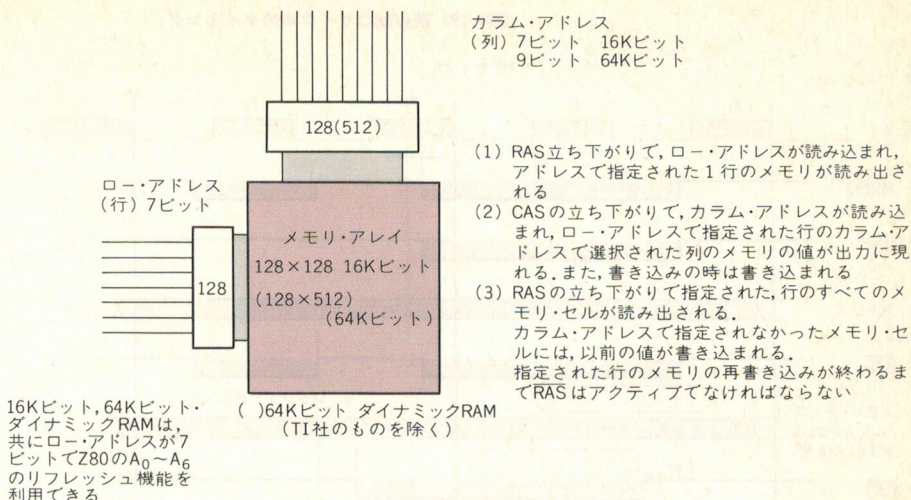
ノイズや各信号のばたつきにより$\overline{RAS}$が立ち下がり，メモリ内のリード/ライトの動作が開始され，完全にメモリ内の動作が完了する以前に$\overline{RAS}$が“H”になると，セルへの書き込み途中で，センス・アンプの信号線がインアクティブになり，メモリへのデータ書き込みが中断し，記憶されている内容が変わる場合があります.

したがって，ダイナミック・メモリのリフレッシュは，CPUのメモリ・アクセスが絶対におきないときに行わなければなりません．Z80は，ダイナミック・メモリのリフレッシュのための信号を，命令の解読のため外部へ動作を行わない M1サイクルのT_3, T_4ステートに出しています.

〈図4-18〉
リフレッシュは，全ロウ・アドレスに行うだけでよい

（1）RAS立ち下がりで，ロー・アドレスが読み込まれ，アドレスで指定された1行のメモリが読み出される

（2）CASの立ち下がりで，カラム・アドレスが読み込まれ，ロー・アドレスで指定された行のカラム・アドレスで選択された列のメモリの値が出力に現れる．また，書き込みの時は書き込まれる

（3）RASの立ち下がりで指定された，行のすべてのメモリ・セルが読み出される．
カラム・アドレスで指定されなかったメモリ・セルには，以前の値が書き込まれる．
指定された行のメモリの再書き込みが終わるまでRASはアクティブでなければならない

そのため，Z80は特別なダイナミック・メモリのコントロール用回路を付加することなく，ダイナミック・メモリを使用したフル・メモリ・システムを実現できます．

最近では，市場にも 256Kビット，1Mビットのダイナミック・メモリがでまわっています．しかし，8ビット・システムの場合は特別なアプリケーションでない限り，64KバイトのRAM領域があればほとんど間に合います．16ビットのCPUの場合は，直接アクセスできるメモリ空間も広く，アプリケーションも画像処理などとなると，いくらメモリがあっても足りないことになり，256Kビットや1Mビットのメモリが大いに利用されることになります．

DRAMの場合，実装密度の向上のためからも16ピ

これだけは　知っておきたい

各言語での演算処理

高級言語での演算処理は，ほとんどが代数とほぼ同様な記述方法で実現できるようになっています．しかしアセンブラでは，これらの処理は簡単な加算であったとしても，数ステップのコーディングを必要とします．また，一度に扱えるデータの大きさも8ないし16ビットであるため，大きな数値を扱う場合，分割して処理しなければならないなど，プログラマに負担がかかります(図4-A参照)．

〈図4-A〉 各種言語での加算例

BASIC，FORTRAN
```
C1＝A1＋B1
  C1は変数
  A1，B1は変数または定数
  ＝は等号でなく代入を示す
```

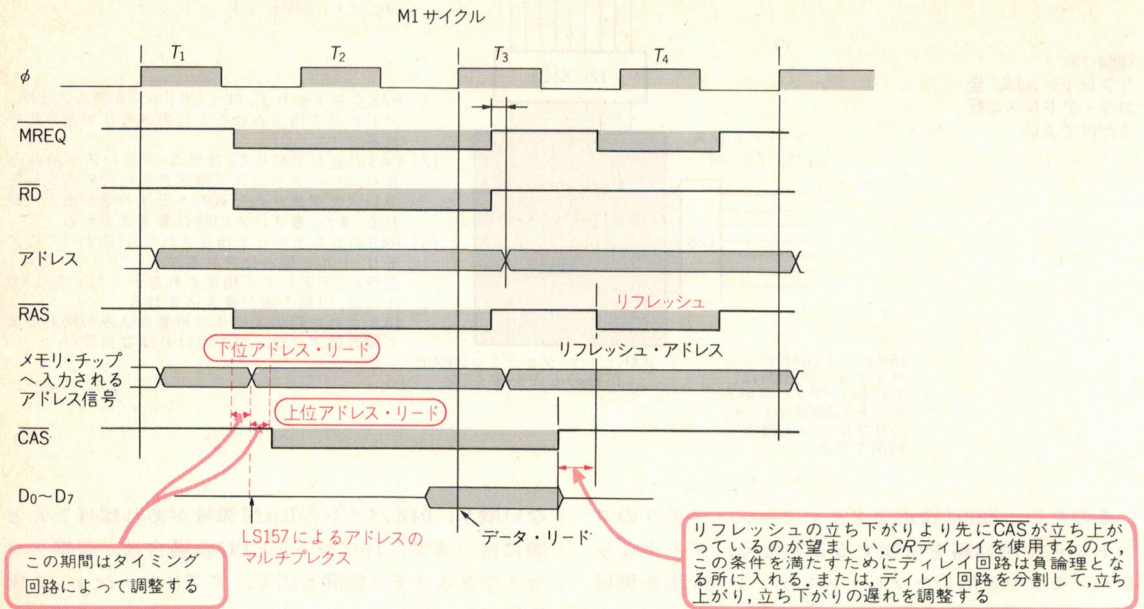
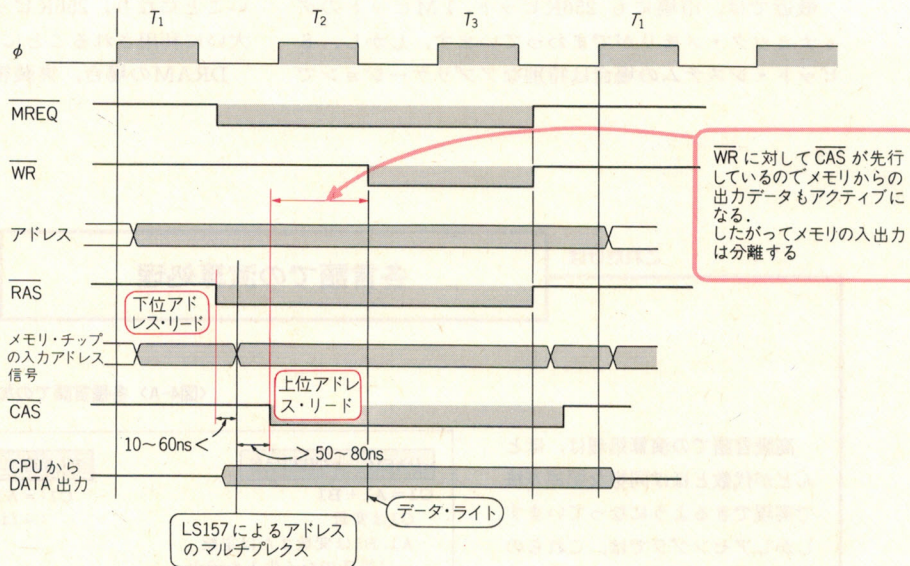
PASCAL
```
C1：＝A1＋B1
  ：＝は代入を等号と区別する
```

Z80アセンブラ
```
LD HL，(A1)
LD BC，(B1)
ADD HL，BC
LD (C1)，HL
   A1，B1，C1はメモリ中の
        16ビット・データ
```

8086アセンブラ
```
MOV AX，A1
ADD AX，B1
MOV C1，AX
   A1，B1，C1はメモリ中の
        16ビット・データ
```

これらアセンブラの記述例は，メモリ中の16ビットの整数データの例で，実数データの場合はこれだけの処理では実現できません．実数演算用のライブラリも用意されていますが，高級言語を使用するのが得策です．

上記の高級言語では浮動小数点の実数データも同様に扱えます．

〈図4-19〉読み出しサイクルのタイミング

〈図4-20〉リード・モディファイ・ライト・サイクル

ンのDIPパッケージのものが，64Kビット，256Kビットともに利用されています．アドレス・ラインは一つの端子がそれぞれの入力タイミングに合わせて，行および列アドレスを入力するようになっています．

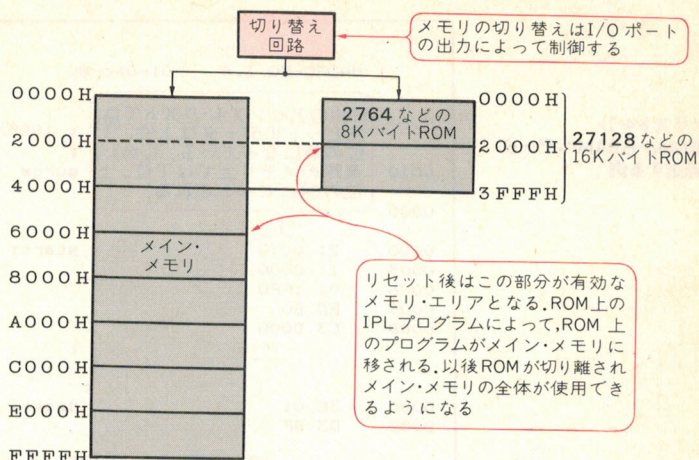
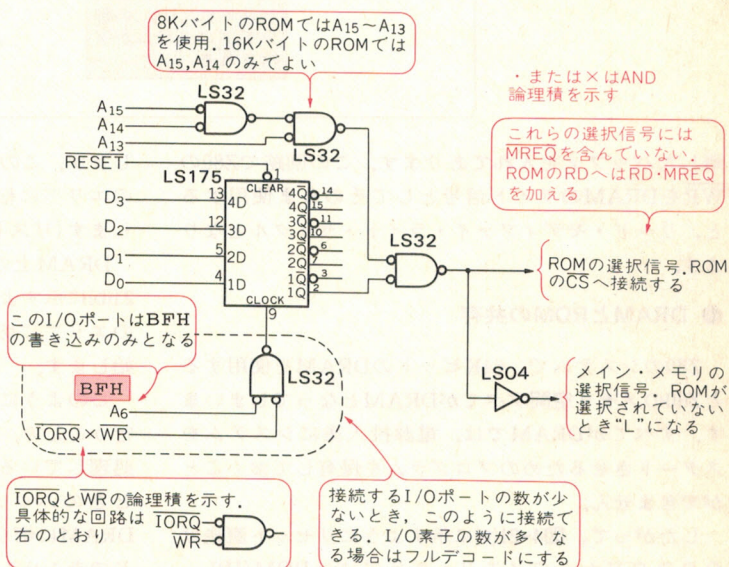
具体的なアドレスの設定および各コントロール・タイミングを図4-19，図4-20に示します．

● DRAMのリード・サイクル

(1) $\overline{\mathrm{RAS}}$信号の立ち下がりで，アドレス入力端子のデータが下位アドレスとして読み込まれる．

(2) $\overline{\mathrm{RAS}}$の立ち下がり後，所定の時間後にアドレス入力端子に加わるデータを上位アドレスに切り替える．この切り替えはDRAMの外で行われる．

(3) アドレス入力が切り替わって，アドレス信号が安定した後，$\overline{\mathrm{CAS}}$信号の立ち下がりで，データが列アドレスとして読み込まれる．この$\overline{\mathrm{CAS}}$信号の立ち下がり後，所定の時間経過後，選択されたアドレスのIC内部のメモリ・セルのデータが出力端子に現れる．

〈図4-21(a)〉
IPL ROMとメイン・メモリの切り替え

〈図4-21(b)〉
IPL ROMとメイン・メモリの切り替え回路の具体的な例

アクセス・タイムは，$\overline{RAS}$または$\overline{CAS}$のそれぞれを起点として，データの確定出力が得られるまでの時間で示されます．

● DRAMでのライト・サイクル

DRAMのライト動作は，$\overline{WR}$信号の立ち下がりのタイミングに応じて，アーリ・ライト・サイクルとリード・モディファイ・ライト・サイクルの二つのモードとなります．

(1) アーリ・ライト・サイクル

一般のメモリと同様にライト動作のみで，出力端子にデータが現れないモードを，アーリ・ライト・サイクルといいます．

このアーリ・ライト・サイクルとなるためには，$\overline{CAS}$の信号の立ち下がり以前にWR信号がアクティブになっている必要があります．$\overline{CAS}$より$\overline{WR}$が先に立ち下がることで，DRAMはこのサイクルがライト・サイクルであることを知ります．ライト・サイクルでは，出力端子にデータを出力する必要がありませんので，出力はハイ・インピーダンスのままです．

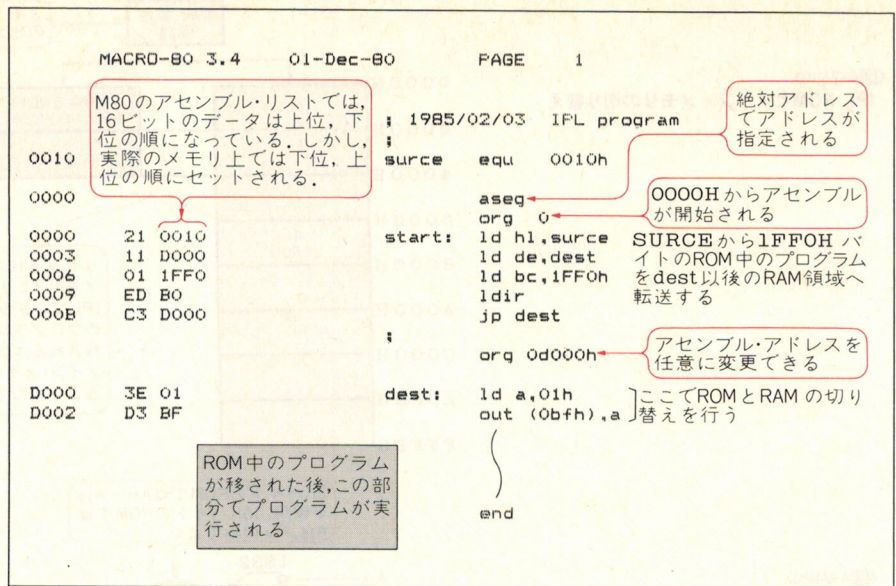
ライト・サイクルがつねにこのアーリ・ライトであることが確実な場合は，入出力信号のDI, DOを直結して配線の数を減らすことができます．

(2) リード・モディファイ・ライト・サイクル

DRAMは，$\overline{CAS}$がアクティブになる前に$\overline{WR}$が“L”になっていないと，$\overline{CAS}$の立ち下がりでリード・サイクルが始まります．

DRAMでは，リード/ライトの各ラインが，別々に端子へ出ています．従って，データ入力端子へ書き込みデータをセットし，$\overline{CAS}$の立ち下がり後ライト・パルスを加えると，出力端子へセルのデータを読み出しながら書き込み処理を行うことができます．これをリード・モディファイ・ライト・サイクルといいます．

図4-3の回路に示すように，ここでは，DI, DOを分

〈リスト4-1〉
ROMからRAMへデータ転送する例

離したバッファを入れてあります．この回路でZ80の$\overline{\text{WR}}$をDRAMのライト信号としてそのまま使用すると，リード・モディファイ・ライト・サイクルとなります．

● DRAMとROMの共存

Z80のシステムで，64KビットのDRAMを使用するとZ80のメモリ空間すべてがDRAMとなってしまいます．すべてがDRAMでは，電源投入後にシステムをスタートさせるためのプログラムを保有しておくことができません．

したがって，図4-21(a)に示すようにリセット直後は０００００Hからのメモリ・エリアにはROM(IPL：Initial Program Loader)がセットされているようにします．このROM上のプログラムをDRAM上の所定のエリアに転送し，制御をDRAMのプログラムに移します(リスト4-1参照)．

DRAM上のプログラムに制御が移った後は，図4-21(b)に示すようにROMのアドレス・エリアをDRAMのエリアに切り替え，全体をDRAMにして処理を開始します．

このように，ROMからDRAMに切り替えて処理を行うことで，システム・クロックを高速バージョンで処理しているときでも，ROMがアクセスされているときのみウェイト・ステートを挿入するだけで，DRAMに切り替えた後はNO WAITの最高のスピードで走らせることができます．

パラレル・インターフェース

■ NEXT

最初に入出力の基本を説明します．そして，8212/8255A/Z80 PIOとの接続例，プログラミングについて説明します．

keywords

PPI：Programmable Peripheral Interface．インテル社のパラレルI/O用のデバイス．8255A．
PIO：Parallel Input/Output Controller．Z80ファミリのパラレル用のデバイス．
双方向ポート：入力，出力いずれのデータの処理機能をももったポート．I/Oポート．
ラッチ：データの変化があっても保持の指定以後は保持されたデータが保存される機能．
8212：汎用のパラレル用のデバイス．プログラマブルでないが多様な使い方が可能．
フラグ・ポート：入力の有無．出力の可否など状態を示すフラグ用のポート．
セントロニクス・パラレル：プリンタ用に一般的に使用されているパラレルのインターフェース．
ハンドシェイク：データ線以外に制御ラインを設けて，データの受け渡しを確実に行うため処理．
ビット・モード：パラレル・ポートをビット単位でON/OFFする機能．各デバイスはこの機能をもっている．

コンピュータ・システムと外部装置のデータの受け渡しを行うために，入出力ポートが必要になります．この入出力ポートには，その機能によっていくつかの仕様があります．また，実現のための方法も何種類かあります．本章では，データ・バスの8ビット全体(もしくはその一部)のデータを同時に処理することを基本とした，パラレルI/Oのインターフェースについて説明します．

入出力の基本機能

入出力ポートとは，コンピュータ・システムと外部の何らかのシステムとの間での，データのやりとりを実現するための仕掛けです．この場合に問題になるのは，外部の装置とCPUシステムとは，通常は同期がとられていず独立に動作していることです．

したがって，ただたんにCPUが入力ポートを読みに行ったとしても，相手側が必ずしもデータを出力しているとは限りません．このような方法では，相手側から順番に一つ一つデータを受信するときなどに，相手側が新しいデータを出力したかどうかの確認を行うことができません．

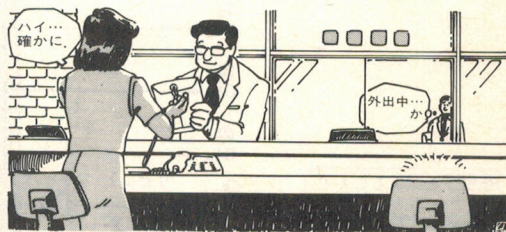
またこちらからデータを相手側に出力したとしても，新しいデータが現在出力されていることを相手に示さなければ，相手側に正しいデータをタイミングよく渡すこともできません．このように，確実なデータの受け渡しが必要な入出力操作の場合，考慮しなければならない問題が多くあります．

● データ入出力方法のタイプ

データ入出力方法にはいくつかの形があります．それらを，次に示します．

(1) 一方的な入力または出力を行う場合

図5-1に示すように，その時点での外部からのデータを，相手の状態と無関係に読み取る場合を考えます．具体的には，相手側の状態を示すフラグのデータを読み込むときなどにこの方法が用いられます．

〈図5-1〉データの受け渡し，入力，出力

また，相手側の状態と無関係にデータを出力する場合もあります．これは，こちら側の状態を相手側に知らせるためのフラグの出力などの場合です．

一般的に，入力はその時点でのデータを読み込めばよいのですが，出力には，次に新たなデータが出力されるまで，出力されたデータを保持しておく機能が必要です．I/O用の周辺装置用のデバイスは，出力として用いた場合にデータを保持するラッチの機能をもっています．

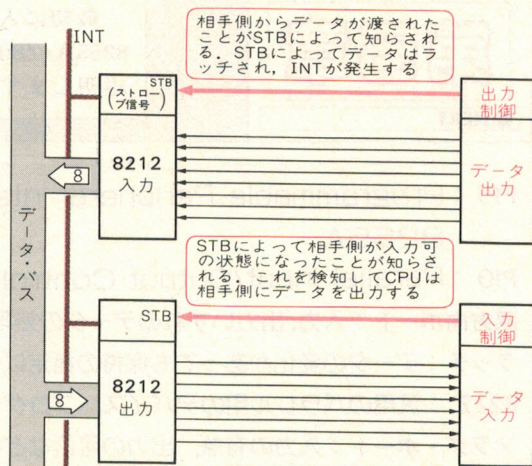
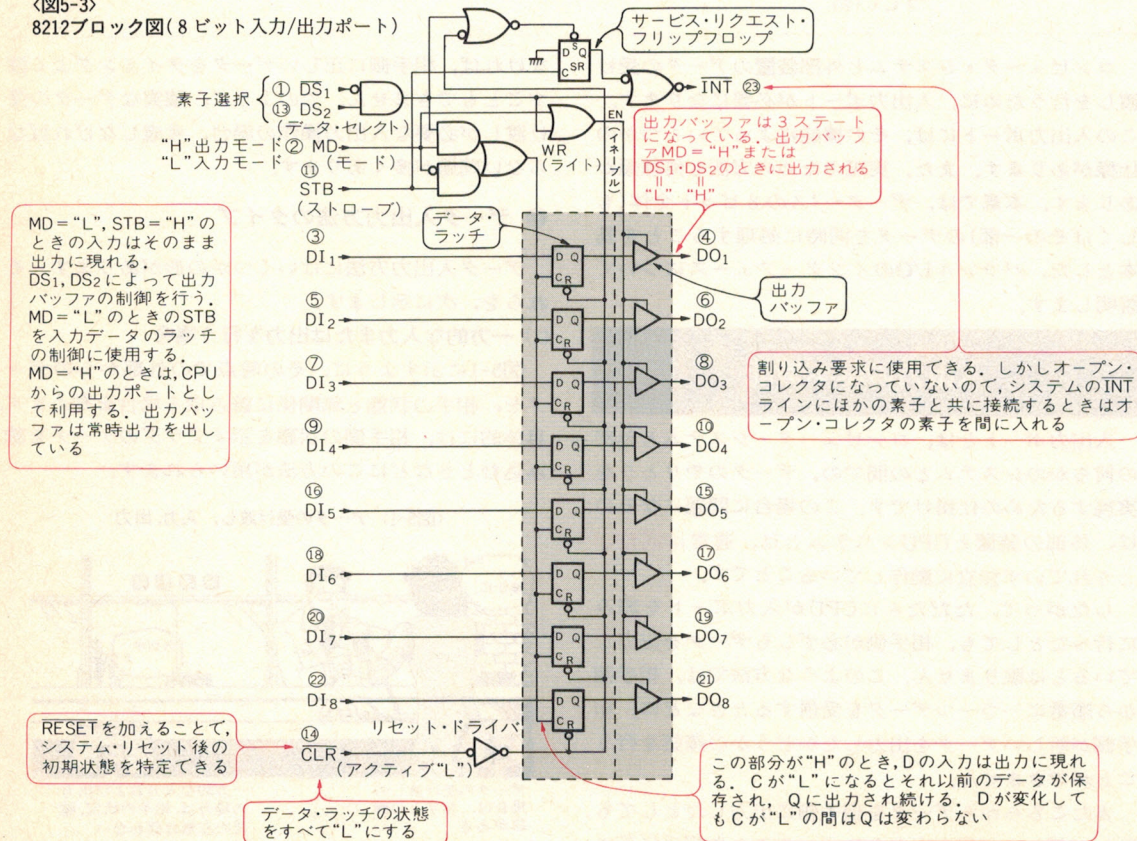
(2) データの受け渡しを行う場合

物の受け渡しは，文字どおり相手が受け取る態勢にあり，手を出していることを確認し，相手の手の内に確実に渡す物を載せます．受け取る側は，相手がすべてを渡し終わったことを確認のうえ，受け取った物を処理します．これと同様なことが，入出力装置でも必要になります．

● データの受け渡し

このように，データの受け渡しを確実に行うためには，データ・ビット以外に入出力の確認を行うための信号線が必要になります．データの処理速度によって，いくつかの場合が考えられます．それについて図5-2に示します．

データ入力の場合，相手側の送信スピードに，こちら側の処理が間に合わない場合が生じます．このときには相手側にこちら側がまだ受信データの処理を終了

〈図5-2〉データの受け渡しを確実に行うハンドシェイク(8212での例)

〈図5-3〉
8212ブロック図(8ビット入力/出力ポート)

しておらず，次のデータが受け付けられない旨を連絡しなければなりません．このために，コントロール線が利用されます．

データ出力の場合は，通常データを保持する機能が必要となります．

CPUからの出力は，OUT命令のときにのみCPUから所定のデータが出力されます．相手側と完全に同期がとれていて，相手側がOUT命令の期間のみで出力されたデータに対する処理を完了するような，特殊なアプリケーションでない限り，データ保持の機能が必要です．

その他に，出力すべきデータを出力ポートにセットした後，相手側に今データが更新されて新しいデータが出力されたことを示す場合もあります．この部分は，いくつかの方法が考えられます．また，この入出力ポートから割り込みが起動される場合もあります．

これら入出力ポート用に，多くの入出力専用のLSIがあります（TTLを使用しても実現できるが，多くの点で専用のLSIを使用したほうが簡単）．

具体的なペリフェラル（周辺）用デバイスの使用法

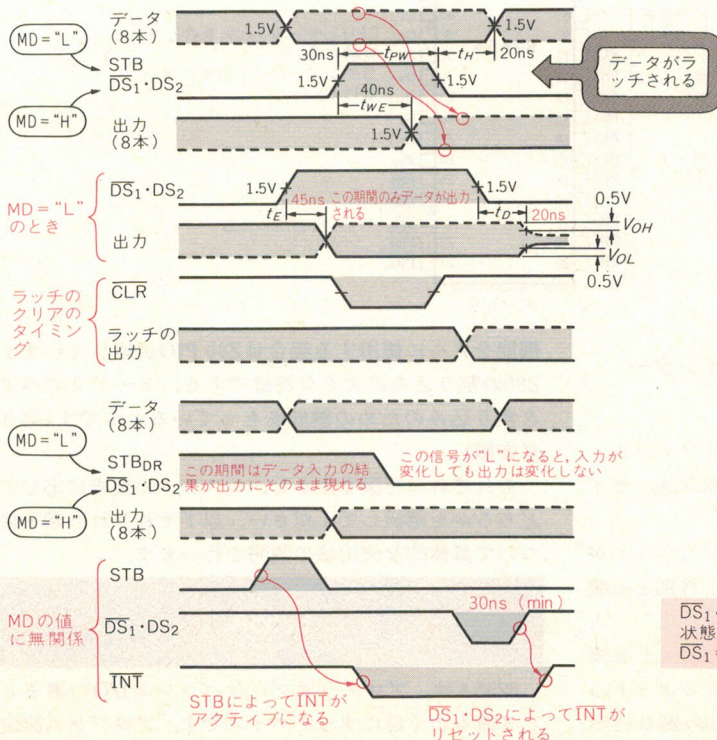
● 8212（図5-3，図5-4）

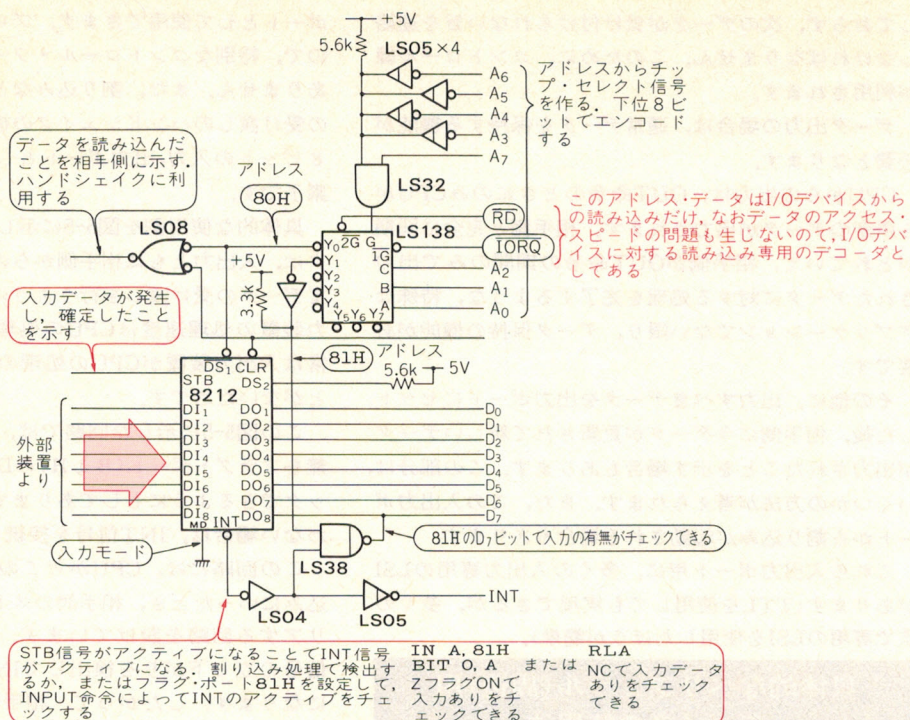
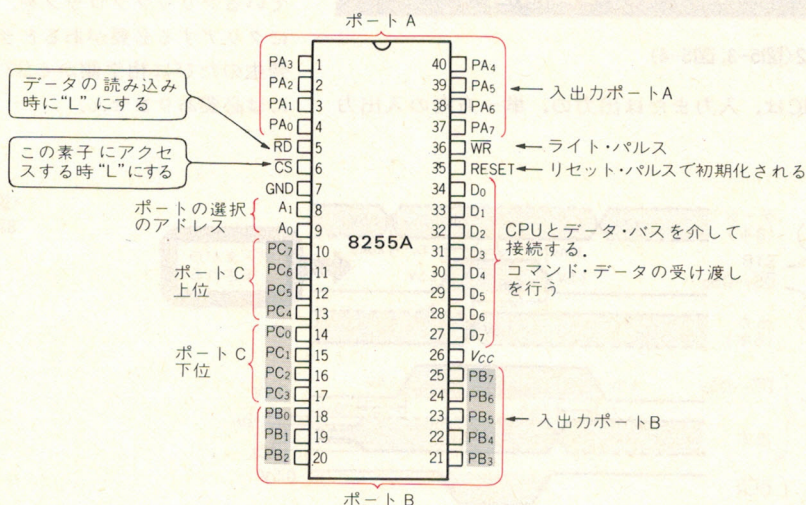
このICは，入力または出力の，単一方向の入出力ポートとして使用できます．プログラマブルではないので，特別なコントロール・ワードを設定する必要はありません．また，割り込みなどの処理およびデータの受け渡しのハンドシェイクの機能ももっているので，8ビットの入力，出力ポートとして手軽に利用できる素子です．

具体的な使用例を図5-5に示します．ハンドシェイクは，入出力ともに相手側からの入出力の要求に応じてデータの受け渡しを行うようになっています．入出力装置の処理速度はCPUの処理速度に比べ遅く，通常は入出力装置がCPUの処理の終了を待つようなことがないためです．

この図5-5に示した回路では，外部からの入力の有無をフラグ・ポート（８１Ｈ）のD_7を調べることでチェックできるようにもしてあります．割り込み処理を行わない場合は，INT信号を接続しないで使用します．

この回路には，CPUからこの素子のデータを読み込みにいったとき，相手側のストローブ・フラグをクリアする回路を設けています．これは，8212のSTB信号の立ち下がりを検出してINT信号がアクティブになります．したがって，相手側のSTB信号を作っているフリップフロップを，データの読み込みと同時にクリアする必要があるときに使用します．データの発生のたびに相手側からSTBパルスが得られる場合には必要ありません．

〈図5-4〉
8212のタイミング

$\overline{DS_1}$・DS_2は，$\overline{DS_1}$がアクティブ，DS_2もアクティブな状態の論理積がアクティブであることを表す．つまり$\overline{DS_1}=1=$"L"かつ$DS_2=1=$"H"のときを示す．

〈図5-5〉
8212を使用した
入力ポートの例

〈図5-6〉
8255Aの端子配置

● プログラマブル・ペリフェラル・インターフェース用のデバイス

Z80に接続できるプログラマブルなペリフェラル・インターフェースとして，インテル社の8255A，ザイログ社のZ80 PIOがあります．

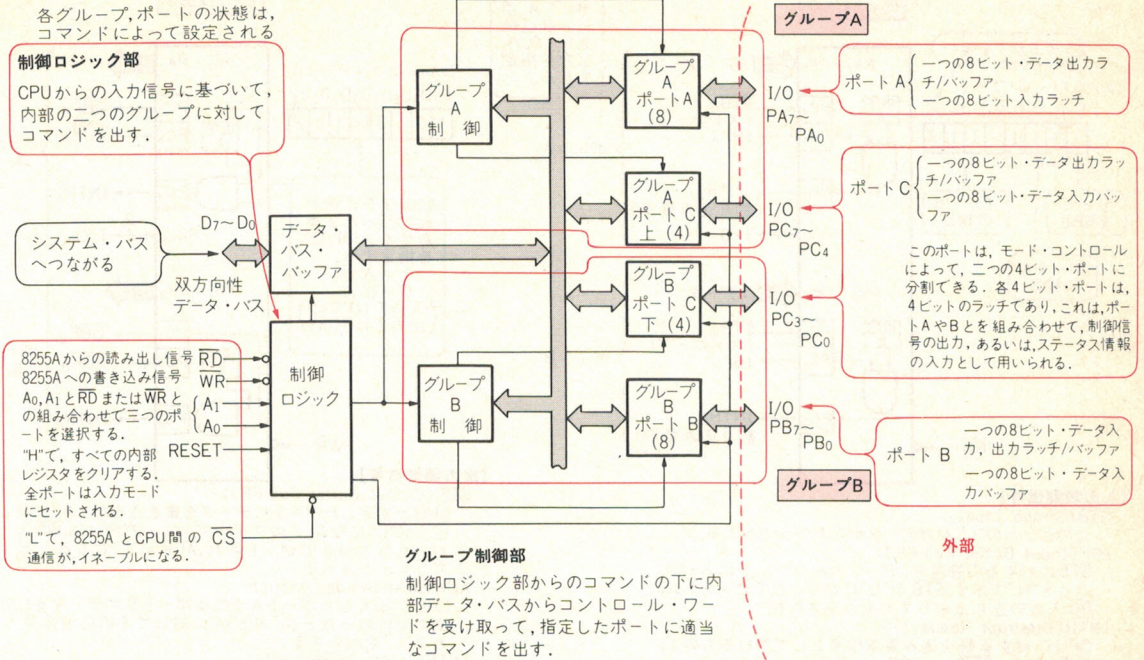
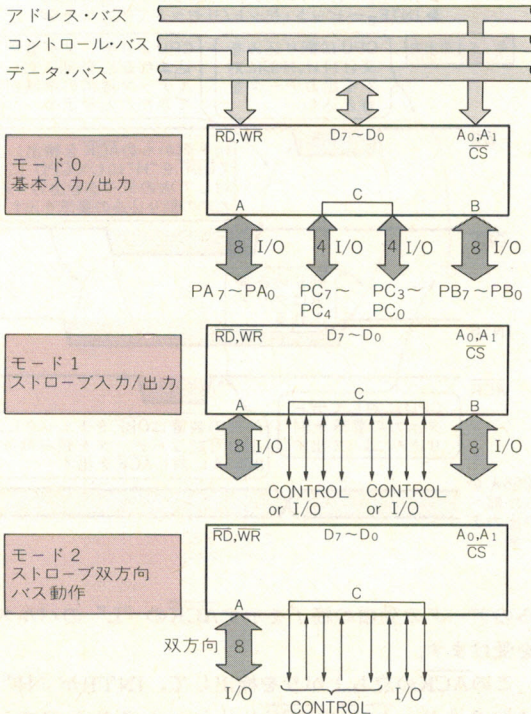
8255Aは，最大3ポート 24ビットの入出力ポートが設定できます．その他に，入出力ポートと外部との間で同期をとることができます．

Z80 PIOは，2ポート 16ビットの入出力ポートを実現することができます．入出力に設定できるポートの数は8255Aに比べて少ないのですが，Z80の割り込み機能をフルに使用する場合はZ80 PIOが適しています．Z80の割り込みの大きな特徴である，モード2のベクタ割り込みのための機能をもっているからです(第8章参照)．

それぞれの特徴を生かし，システムの要求に応じてどちらかを選択してください．以下それぞれの素子について具体的な使用法の説明を行います．

8255A

8255Aは，プログラマブルなパラレルI/Oの素子として最もよく目にするデバイスです．プログラム設定

〈図5-7〉 8255Aの内部ブロック図

〈図5-8〉 8255Aの基本モード

によって多くの応用範囲があります．最初は，応用範囲の広いこのようなデバイスを，確実に理解することから始めるのがよいでしょう．

8255Aの端子配置図を図5-6に，図5-7にブロック図を示します．

この素子にはI/OポートとしてA, B, Cの三つがあります．A, B, Cと三つに分けて使う場合と，AとCの半分，BとCの半分という，二つのグループに分けて考えることもできます．

図5-8に示すように機能別にモードが三つ用意されていて，各モードに応じて，A, Bはそれぞれ独立して機能します．Cは，モード0以外は4ビットずつA, Bのそれぞれのグループとともに，コントロール信号としての機能を果たします．

(1) モード0は，たんなる入出力ポート

このモードでは，A, BおよびCの半分ずつを任意にそれぞれを入力または出力ポートとして設定できます．この場合，入力動作ではその入力時にA, B, Cの各ポートに加わっているデータがCPUに読み込まれます．出力動作では，各ポートへ出力されたデータは，出力動作後も同じデータが出力され続けます．これは，各ポートへの書き出し時に出力バッファにデータが保持されるためです．

(2) モード1ではストローブ入出力を行う

図5-9に示すように，A, Bポートをそれぞれ入力と出力の任意に設定してCポートのラインを使用し，ハンドシェイクを行います．

入力動作では，$\overline{STB}$入力を受けて，入力データを入力ポートにラッチします．$\overline{STB}$に合わせて8255Aは

〈図5-9〉 8255Aモード1の説明

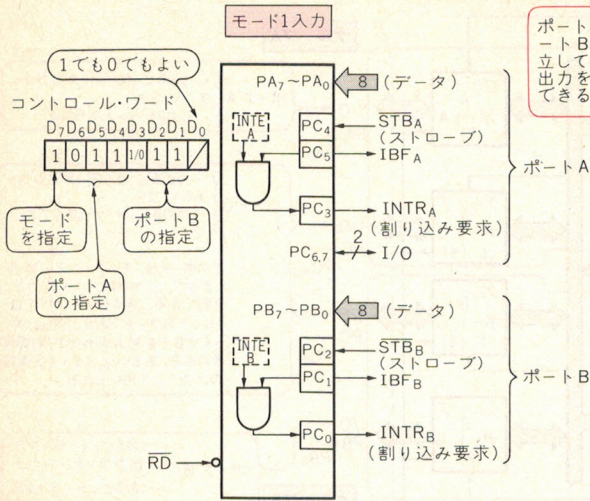
ポートA,ポートBは独立して入力,出力を指定できる

モード1出力

コントロール・ワード
$D_7D_6D_5D_4D_3D_2D_1D_0$
1 0 1 0 1/0 1 0

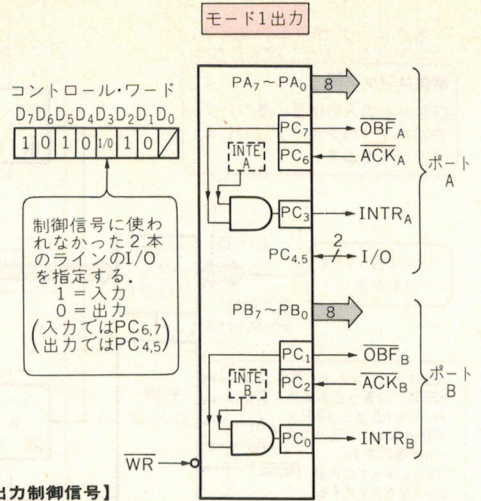
制御信号に使われなかった2本のラインのI/Oを指定する.
1＝入力
0＝出力
(入力では$PC_{6,7}$
出力では$PC_{4,5}$)

$PA_7 \sim PA_0$ 8　PC_7 $\overline{OBF}_A$　INTE A　PC_6 $\overline{ACK}_A$　PC_3 $INTR_A$　$PC_{4,5}$ 2 I/O　ポートA

$PB_7 \sim PB_0$ 8　PC_1 $\overline{OBF}_B$　INTE B　PC_2 $\overline{ACK}_B$　PC_0 $INTR_B$　ポートB　$\overline{WR}$

【入力制御信号】

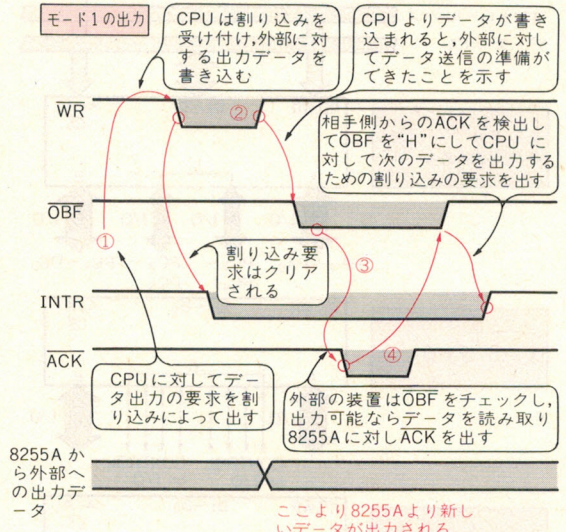
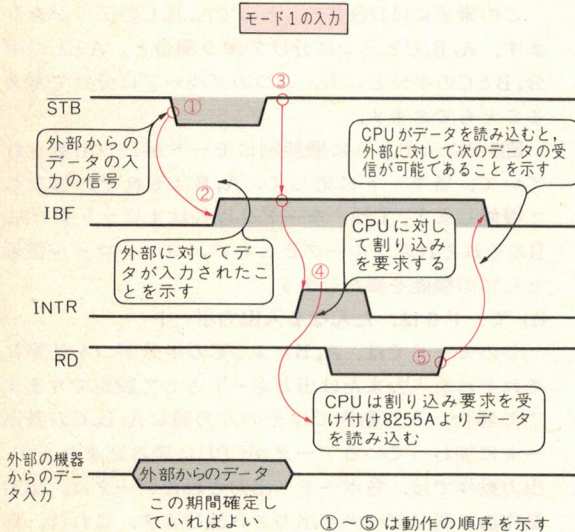
$\overline{STB}$(Strobe Input):
このピンへの"L"入力データが入力ラッチへ入る.

IBF(Input Buffer Full FF):
$\overline{STB}$に対する応答信号.データが入力ラッチに入ったことを"H"で示す.$\overline{STB}$が"L"になることでセットされ,$\overline{RD}$入力の立ち上がりでリセットされる.

INTR(Interrupt Request):
CPUに対する割り込み要求信号として用いるもので,"H"になる.$\overline{STB}$＝1,IBF＝1,INTE＝1のときセットされ,$\overline{RD}$の立ち下がりでリセットされる.
内部割り込みマスク・フリップフロップの制御は,
▶$INTE_A$ ―ビット・セット/リセットによるPC_4の制御
▶$INTE_B$ ―ビット・セット/リセットによるPC_2の制御

【出力制御信号】

$\overline{OBF}$(Output Buffer Full FF):
CPUが指定したポートにデータを書き込んだとき,このピンが"L"になる.このフリップフロップ(FF)は,$\overline{WR}$入力の立ち上がりでセットされ,$\overline{ACK}$が"L"になることでリセットされる.

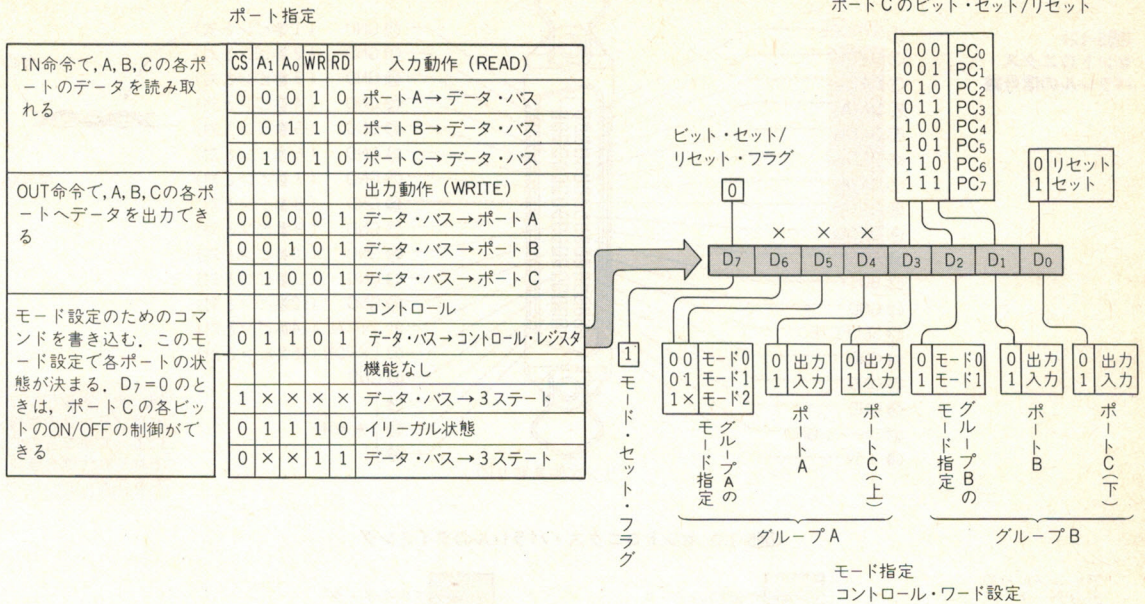
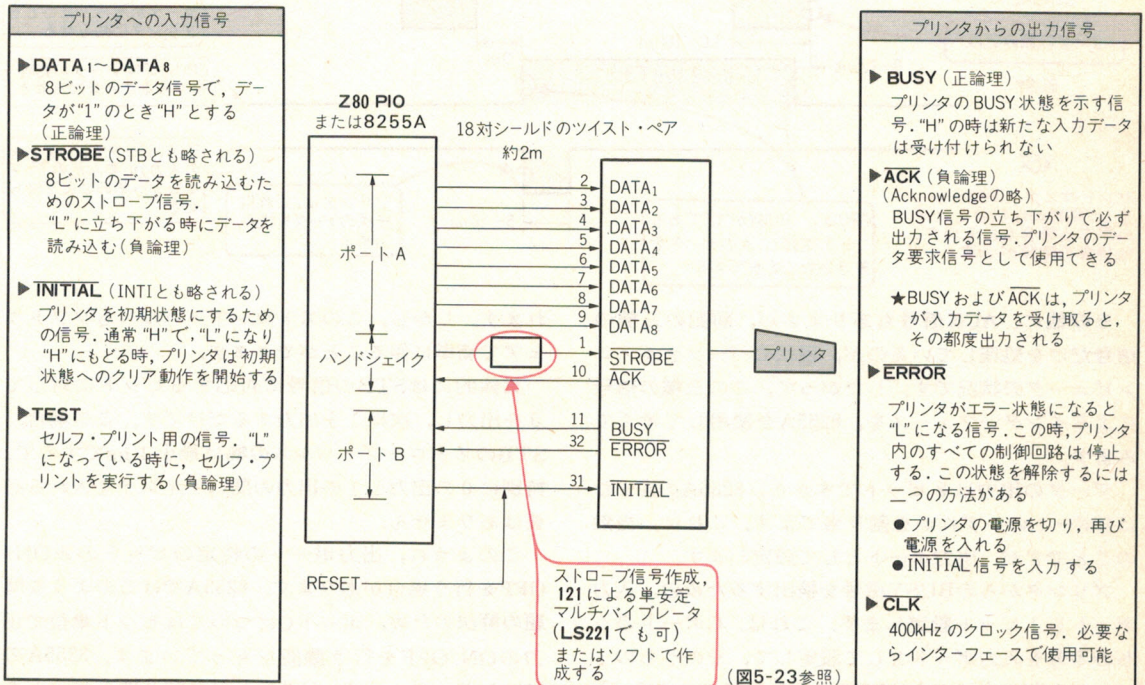
$\overline{ACK}$(Acknowledge Input):
CPUが出力したポートAまたはポートBのデータをI/O側で受け取ったとき,8255Aに対してその応答信号である"L"を出力する.

INTR(Interrupt Request):
CPUに対する,I/Oからのデータ転送終了割り込み信号で"H"が出る.この信号は,$\overline{OBF}$＝1,INTE＝1のときに,$\overline{ACK}$によってセットされ,$\overline{WR}$の立ち下がりでリセットされる.
内部割り込みマスク・フリップフロップの制御は,
▶$INTE_A$ ―ビット・セット/リセットでPC_6の制御
▶$INTE_B$ ―ビット・セット/リセットでPC_2の制御

相手にIBFを出します．このIBFはCPUが入力データを読み取るとOFFになります．IBFはデータが未処理で次のデータが受信できないことを示します．

出力動作時は次のようになります．データを出力するライト・パルスの立ち下がりでINTRが“L”になります．出力データが出力ポートに確定すると，I/O側へ$\overline{OBF}$を“L”にして知らせます．そして，I/O側からのデータの受信の終了を示す$\overline{ACK}$の“L”のパルスを受けます．

この$\overline{ACK}$の立ち上がりを検出して，INTRが“H”になります．$\overline{ACK}$で$\overline{OBF}$もリセットされるので，INTR，$\overline{OBF}$によってもI/O側の受信終了を知ることができます．

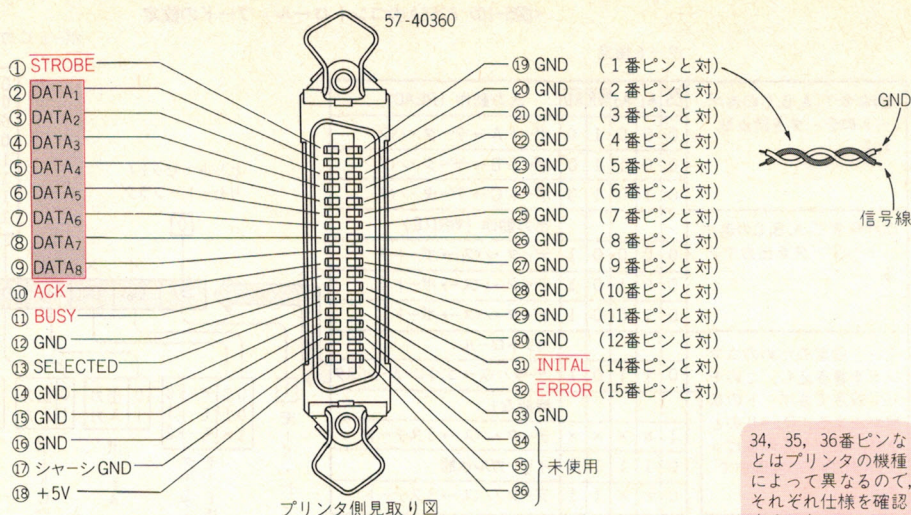
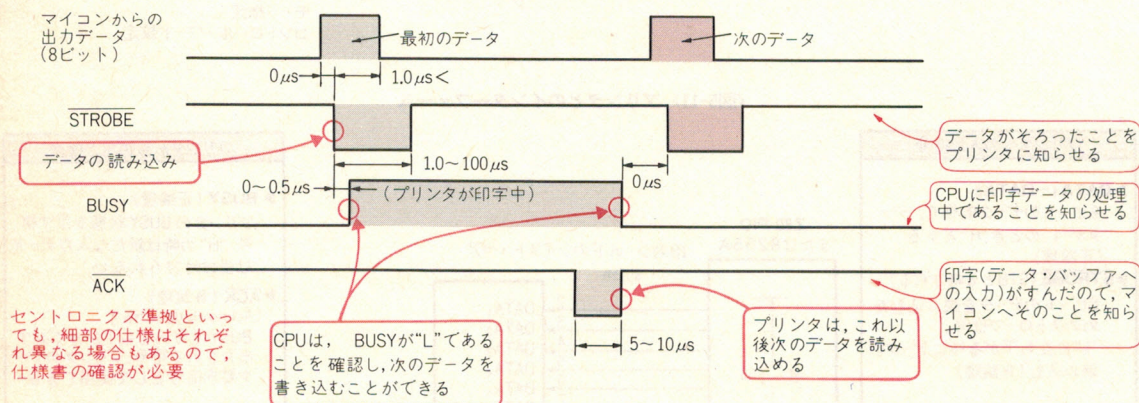
モードの設定とか，入出力方向を決めるコントロー

〈図5-10〉 8255Aのコントロール・ワードの設定

〈図5-11〉 プリンタとのインターフェース

ル・ワードの設定を図5-10に示します.

● プリンタのインターフェースに用いて説明

このモードでの実例として, プリンタのインターフェースを考えてみます. このプリンタのインターフェースとしては, 現在一般的にはセントロニクス・パラレルと呼ばれている仕様のものが用いられています.

これは, 図5-11に示すようにプリンタへの出力データ8ビット, プリンタがデータを受信可能かどうかを示すBUSY信号(この信号は, プリンタが読み込んだデータの処理またはプリンタ中で, 次のデータを受け付けられないことが生じていることを示す), プリンタにデータを渡すきっかけを作るSTB信号より成ります(図5-12, 図5-13).

〈図5-12〉
セントロニクス・パラレルの信号線

〈図5-13〉 セントロニクス・パラレルのタイミング

その他に，ACK信号もありますが，前記の三種の信号だけを処理しているのが，現状のパーソナル・コンピュータの状況です．したがって，この三種の信号によるインターフェースを，8255Aを使用して考えてみます．

データの出力は8ビットですから，8255AのAまたはBポートのいずれかを割り当てます．これは，当然のことですが，出力ポートとして設定します．

プリンタからのBUSY信号を検出するための，入力ポートを1ビット設定します．これは，Cポートのうちの半分を入力ポートとして設定して，そのうちの1ビットを割り当てます．STB信号としては，Cポートの残り半分を出力ポートとして設定し，そのうちの1ビットを当てます．

具体的な回路図を図5-14に示します．アドレス・デコーダおよびデータ・バスのバッファなどは，必要に応じて入れます．

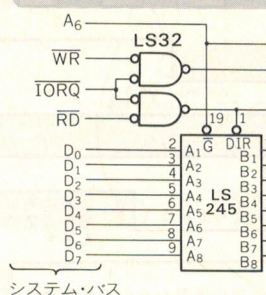
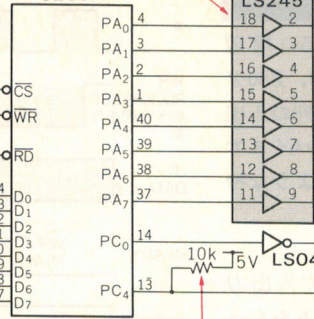
ストローブ・パルスは，ハードウェアでワンショット・マルチバイブレータなどを使用することも考えられます．しかし，このストローブ・パルスもソフトウェアで簡単に作ることができます．

具体的にはSTBの信号に対応するビットに対して0を出力し，次に1を出力するだけです．この場合，STBのストローブ・パルスの幅は最小1μsですので，特別に0の出力と1の出力の間にディレイを入れる必要はありません．

このように，出力ポートの特定のビットのみON/OFFを行う場合があります．8255Aではこのような問題の解決のため，ポートCについてはビット単位で出力のON/OFFを行う機能をもっています．8255Aのコマンド・ポートに，Cポートの目的とするビットを制御するコマンドを書き込むことで実現できます．

図5-14の回路に対するプログラムをリスト5-1，リスト5-2に示します．それぞれ，アセンブラ，ターボ・パスカルの各言語で記述してあります．

8255Aは出力ポートでも，そのポートを読み込み，その時点で出力されているデータを得ることができます．この機能を利用して，次の方法で特定のビットを

〈図5-14〉
8255A によるセントロニクス・パラレル・インターフェース例(プリンタ用インターフェース)

〈リスト5-1〉 8255Aを用いたアセンブラによるプリンタ制御例

```
MACRO-80 3.4    01-Dec-80       PAGE    1

                        ; program Z80501
                        ; 85/02/09 Y.Kanzaki code
                        ;
0090                    basepi  equ     090h            ; 8255A  port address
0090                    ppia    equ     basepi          ; 8255A  ppi A port
0091                    ppib    equ     basepi + 1      ; 8255A  ppi B port
0092                    ppic    equ     basepi + 2      ; 8255A  ppi C port
0093                    ppicm   equ     basepi + 3      ; 8255A  ppi cmd
                        ;
0096                    cmd1    equ     096h            ; A,CL in other out
                        ;
0009                    setstb  equ     009h            ; strobe set
0008                    resetstb  equ   008h            ; reset strobe
0004                    buzy    equ     04h             ; list busy
                        ;
                                org
0000'   3E 96           initpi: ld A,cmd1               ;
0002'   D3 92                   out (ppic),A            ;
0004'   C9                      ret
                        ;
0005'                   list_stat:
0005'   DB 92                   in A,(ppic)             ;
0007'   E6 04                   and buzy                ;
0009'   C8                      ret Z
000A'   3E FF                   ld A,255                ;   255=0FFH
000C'   C9                      ret
                        ;
000D'   CD 0005'        list:   call list_stat          ;
0010'   20 FB                   jr nz,list              ;
                        ;
0012'   79                      ld A,C                  ;
0013'   D3 91                   out (ppib),A            ;
0015'   3E 09                   ld A,setstb             ;
0017'   D3 93                   out (ppicm),A           ;
0019'   3E 08                   ld A,resetstb           ;
001B'   D3 93                   out (ppicm),A           ;
001D'   C9                      ret
                        ;
                                end
```


〈リスト5-2〉 8255Aを用いたパスカルによるプリンタ制御例

```
 1: {  PPI test program  }
 2: {  1985/02/09        }
 3: {  code Y.Kanzaki    }
 4: program Z80501;
 5:
 6: procedure ppi_init;
 7: const
 8:   ppicm  = $93;
 9:   cmd1   = $96;
10: begin
11:   port[ ppicm ] := cmd1;
12: end;
13:
14: function lstat : boolean;
15: const
16:   ppic = $92;
17:   buzy = $04;
18: begin
19:    lstat := (port[ ppic ] and buzy)=0;
20: end;
21:
22: procedure list( data : byte );
23: const
24:   ppib   = $91;
25:   ppicm  = $93;
26:   setstb  = $09;
27:   resetstb = $08;
28: begin
29:   repeat
30:     until lstat;
31:   port[ ppib ] := data;
32:   port[ ppicm ] := setstb;
33:   port[ ppicm ] := resetstb;
34: end;
35:
36:
37:
38: begin
39:    ppi_init;
40:    list( $31 );
41:    list( $0d );
42: end.
```

ON/OFFすることもできます.

(1) 現在の出力の状態を得る.出力ポートを読み込むか,つねに出力データのコピーをメモリ中に保存しておく

(2) (1)で得られたデータの必要とするビットを,セットまたはリセットする.この処理には,ビット操作命令,または論理演算命令を使用する

(3) (2)で処理した結果を出力する.

以上の操作で所定のビット以外を変化させることなく,目的のビットのON/OFFができます.

具体的な使用例は次のようになります.

(例)
```
IN   A, (PPIC)
RES 0, A
OUT  (PPIC), A
SET 0, A
OUT  (PPIC), A
```

これは,プリンタ・インターフェースの$\overline{\text{STB}}$の出力例です.リスト5-1を参照してください.

Z80 PIO(Parallel I/O)

前項の8255Aは8080A/85の周辺LSIでしたが，Z80 PIOは，Z80ファミリの汎用の８ビット並列の入出力インターフェース用のデバイスです．このデバイスは，次のような特徴をもっています．

▶40ピンDIPである

▶A, B 二つの，８ビットで入出力を任意に設定できるポートをもっている

▶各ポートは次の四つのモードに設定できる(図5-15, 図5-16, 図5-17参照)．

0：出力モード

1：入力モード

2：双方向モード

3：ビット・モード

▶ハンドシェイクのための信号線を二つもっている

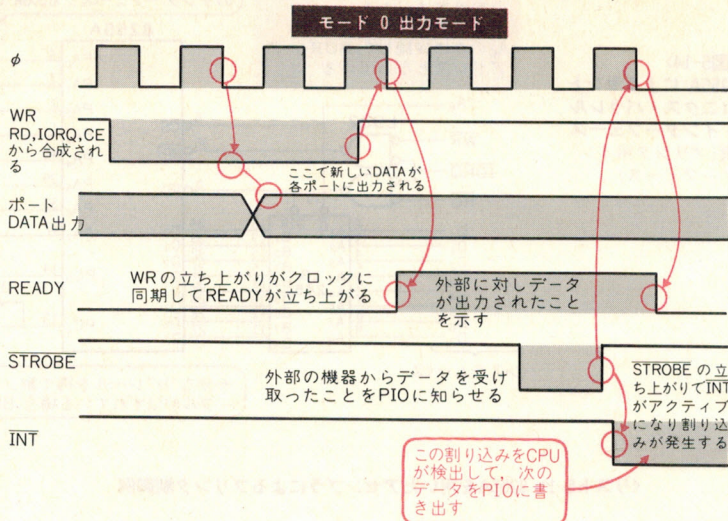
▶Z80のモード２の割り込みのための，ベクトルの生成機能およびデイジィ・チェインの割り込みの処理機能を内蔵している

▶すべての入出力ラインは，TTLコンパチブルとなっている．またポートBは，直接ダーリントン・トランジスタなどの電流容量の大きい負荷をドライブする能力をもっている．

Z80独自の機能をフルに利用しようとするとき，Z80ファミリの周辺装置用のデバイスを使用すると，特別なハードウェアの追加もなく構成することができます．

Z80 PIOのブロック図を図5-18に，各端子の機能を図5-19に示します．Z80 PIOはコマンドの設定によって，初めてI/Oデバイスとしての機能を発揮します．このコマンドの設定には，A, Bの各ポート用のコマンド・ポートが用意されています．

これはA, Bの各ポートの選択用の入力端子(６番ピン)と，それぞれのポートのコマンドであるかデータであるかを選択する入力端子(５番ピン)の二つによって各ポートの選択が制御されます．具体的にはこれらの入力端子にアドレス・バスのA_0, A_1を接続すると，図5-20に示すようにそれぞれに対応したポート・アドレスが設定できます．

〈図5-12〉 Z80 PIOモード０のタイミング図

〈図5-16〉 Z80 PIOのモード１のタイミング図

〈図5-17〉 Z80 PIOのモード３のタイミング図

PIOのリセット

M1端子(37)に，システム・バスのM1と，RESET(システム・バスのRESETを反転させる)のORを加えると，RESETで，PIOも同時にリセットすることができる．RESETは，通常マシン・サイクルに対して十分長い期間"L"になる

〈図5-20〉 Z80 PIOとA_0, A_1の接続例(図5-23参照)

	A_1 C/D	A_0 A/B
Aポート・データ	0	0
Bポート・データ	0	1
Aポート・コマンド	1	0
Bポート・コマンド	1	1

データ，コマンド・ポートが連続してアドレスとなる

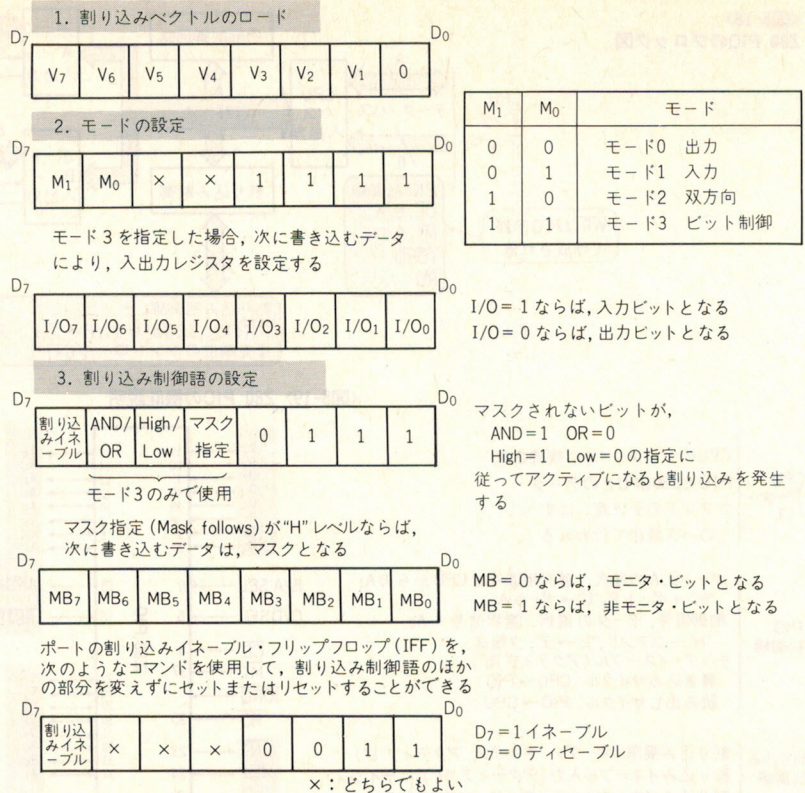
	A_1 A/B	A_0 C/D
Aポート・データ	0	0
Aポート・コマンド	0	1
Bポート・データ	1	0
Bポート・コマンド	1	1

それぞれポートのデータ，コマンドが連続したアドレスとなる

8255Aでもハンドシェイクの必要な場合は，A, Bの2ポートしか使用できません．その場合，割り込み処理の機能を内蔵したZ80 PIOのほうが有利になります．

● Z80 PIOの初期設定は数ステップのコマンド設定を必要とする

Z80 PIOの初期設定はA, Bの各ポートに対して，必

〈図5-21〉 Z80 PIOの制御語

これだけは　知っておきたい

オープン・コレクタ

各デバイスの出力端子の中に，オープン・コレクタ(TTL)または，オープン・ドレイン(FET)と呼ばれるものがあります．割り込み関係(INT)がワイヤードORされるために，そうなっています．

これは，図5-Aに示すように最終の出力段のトランジスタのコレクタがそのまま出力端子に出ています．この出力段のトランジスタは，出力が"H"のときカットオフ(コレクタ-エミッタ間に電流が流れない)，出力が"L"のときはON(コレクタ-エミッタ間に電流が流れる)になります．

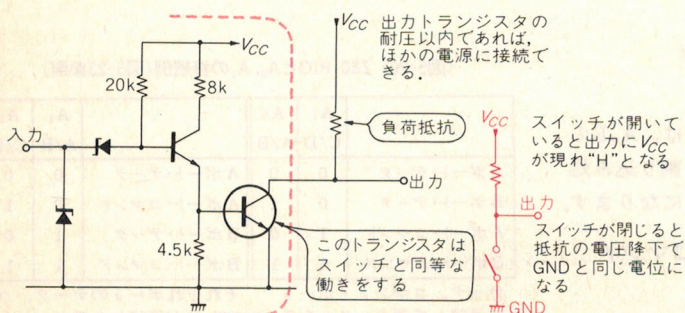
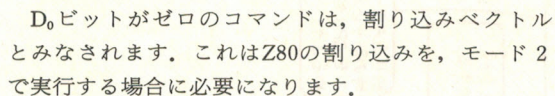
この出力端子と電源とを負荷抵抗で結び，出力を電圧のレベルとして取り出すことができます．この場合の電源電圧は，出力段の素子の耐圧内で自由に選ぶことができます．したがって，異なった電圧レベルの変換などにも利用することができます．もう一つのオープン・コレクタの重要な利用法に，ワイヤードNORとしての利用法があります．これは複数の出力を共通の負荷抵抗で接続しておきます．これにより，すべての出力が"H"のときのみ，共通に接続された出力は"H"となります．共通に接続された出力のうち一つでも"L"となったなら，出力は"L"となります．このようにオープン・コレクタの素子は，出力同士を接続してもレベルの異なる出力が影響して出力が不安定になったり，出力電流が増大して素子を破壊するようなことはおきません．

〈図5-A〉 出力の構成

〈図5-22〉 Z80 PIOのコマンド設定のフローチャート

要に応じて三種類のコマンドを書き込むことで行います．その三種類のコマンドは，次に示すものです(**図5-21**)．

(1) 割り込みベクトル

D_0ビットがゼロのコマンドは，割り込みベクトルとみなされます．これはZ80の割り込みを，モード2で実行する場合に必要になります．

(2) モード設定

Z80 PIOで設定可能な四つのモードを指定するためのコマンドです．D_7, D_6ビットの組み合わせで，図に示すように0から3までのモードが決まります．このコマンドはD_0からD_3の四ビットがともに1となっています(**図5-22**)．

モード3のビット制御モードを指定した場合は，次に各ビットの入出力を決めるためのコマンドを書き込みます．各ビットは1で入力，0で出力になります．

(3) 割り込み制御のコマンド

PIOからの割り込み要求の可否を制御するコマンドです．ビット制御モードに対しては，各ビットごとに割り込みの必要の有無を指定することができます．また，割り込みの発生するための条件を，ビット同士のOR，またはANDの関係からも指定することができます．

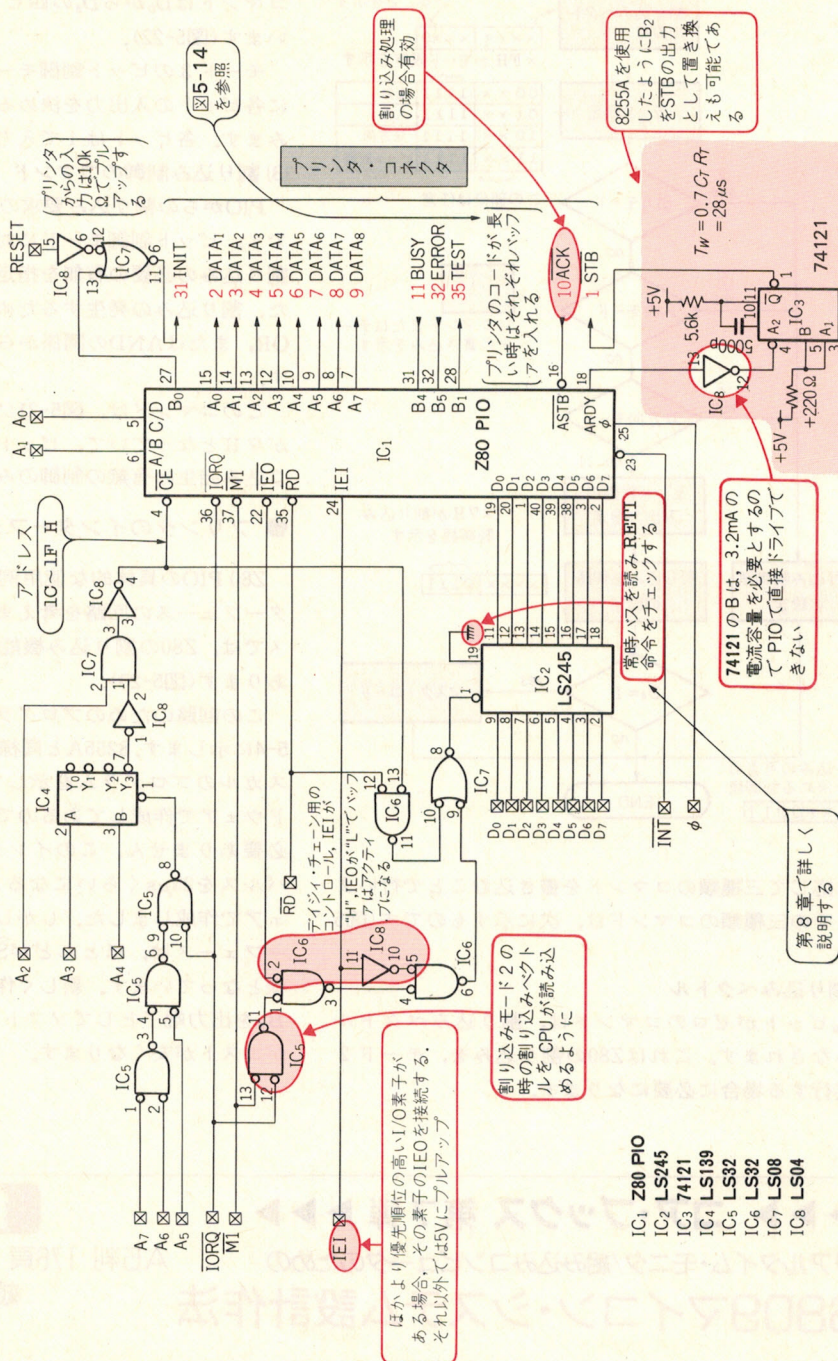
このコマンドは，**図5-21**に示すように下位4ビットが7Hとなっていて，ビット制御モード以外では，割り込み発生の有無の制御のみを行います．

● プリンタのインターフェースに用いて説明

Z80 PIOの具体的な使用例として，プリンタのインターフェースの回路を考えます．このインターフェースでは，Z80の割り込み機能を利用できるようにしてあります(**図5-23**)．

この回路のためのプログラムを，**リスト5-3**, **リスト5-4**に示します．8255Aと同様にアセンブラ，ターボ・パスカルのプログラムを示してあります．$\overline{STB}$はハードウェアで作成してあるので，ソフトウェアの処理は必要ありません．このインターフェースは，$\overline{STB}$のパルスを25μsくらいになるようにするためハードウェアで作成しました．しかし最近のプリンタのインターフェースは，ほとんどが$\overline{STB}$パルスは約1μsの仕様となっています．新しく作るなら図に示すように，B_2を出力端子としてソフトウェアで$\overline{STB}$を作る方法がコストが安くなります．

〈図5-23〉 Z80 PIOを用いたプリンタ・インターフェース

〈リスト5-3〉Z80 PIOを用いたアセンブラによるプリンタ制御例

```
MACRO-80 3.4    01-Dec-80       PAGE    1

Z80 PIOによるプリンタ・イン          ; program Z80502.MAC
ターフェースの処理プログラム          ; Z80-PIO test routine
                                      ; 85/02/10 Y.Kanzaki
                                      ;
001D          Z80 PIOの各ポート ┌piopac  equ  01Dh      ; PIO port address
001C          のアドレス        │piopad  equ  01Ch      ;
001F                            │piopbc  equ  01Fh      ; PIO port address
001E                            └piopbd  equ  01Eh      ;
000F  Z80 PIOモード0の設定コマンド  cmd_out equ  00Fh      ; PIO
00CF  Z80 PIOモード3の設定コマンド  cmd_bit equ  0CFh      ; PIO
00F0  ビット制御モードでのI/O設定   cmd_iO  equ  0F0h      ; in D7-D4 out D3-D0
                                      ;
                                              org
0000'   3E 0F  Z80 PIOのモード    pioinit:ld   A,cmd_out        ;┐ポートAをモード0に設定
0002'   D3 1D  設定ルーチン               out  (piopac),A       ;┘
0004'   3E CF                             ld   A,cmd_bit        ;┐ポートBをビット制御モー
0006'   D3 1F                             out  (piopbc),A       ;│ドにして,各ビットの入出
0008'   3E F0                             ld   A,cmd_io         ;┘力を設定
000A'   D3 1F                             out  (piopbc),A       ;
000C'   C9                                ret
                                      ;
000D'   DB 1E  プリンタの状況の   lstat:  in A,(piopbd)         ;
000F'   CB 67  チェック                   bit 4,A               ; プリンタが受信可能でな
0011'   3E FF                             ld A,0ffH             ; ければA=0FFHとして
0013'   C8                                ret Z                 ; もどる.
0014'   AF                                xor A                 ; プリンタが受信可能なら
0015'   C9                                ret                   ; A=0
               プリンタへの出力
               ルーチン                ;
0016'   CD 000D' STBはZ80 PIO よ list:   call lstat          ;┐プリンタが受信可能にな
0019'   B7       り出力される             or A                ;┘るまでループする
001A'   28 FA                             jr z,list           ;
001C'   79                                ld A,C              ;┐Cレジスタにセットされた
001D'   D3 1C                             out (piopad),A      ;│データを,プリンタに出力
001F'   C9                                ret                 ;┘する
                                      ;
                                          end                 ;
```

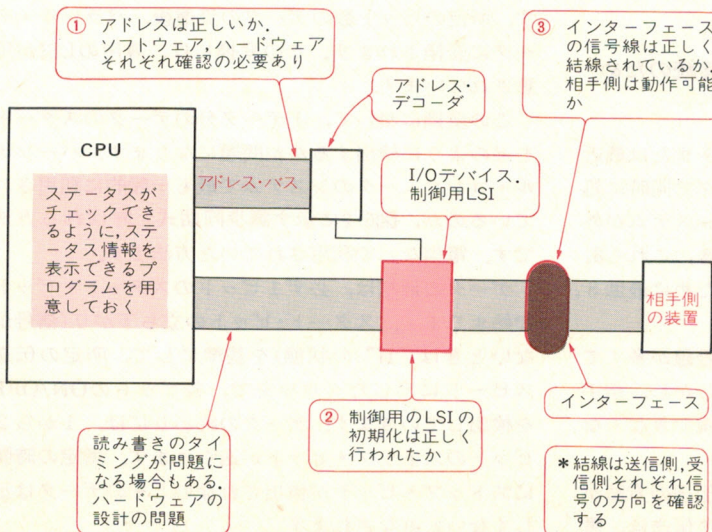
〈リスト5-4〉Z80 PIOを用いたパスカルによるプリンタ制御例

```
{ Z80-PIO test program  }
{ 1985/02/10 code Y.Kanzaki }
program piotest;
const
  piopac = $1D; ┐Z80 PIOの各ポート・アドレ
  piopad = $1C; │スを定数として定義する
  piopbc = $1F; │
  piopbd = $1E; ┘
  cmd_out = $0F; ┐モード制御コマンドを定数
  cmd_bit = $CF; │として定義する
  cmd_io  = $F0; ┘

procedure pioinit;
begin                                   ┐Z80 PIOの
  port[ piopac ] := cmd_out;            │モード設定
  port[ piopbc ] := cmd_bit;            │を行ってい
  port[ piopbc ] := cmd_io;             ┘る
end;

function lstat:boolean;     プリンタの状態
begin                       チェックの関数
  lstat := (port[ piopbd ] and $10)= 0;
end;

procedure list( data : byte );
begin
  repeat                  ┐プリンタが受信可になるまでまって
    until lstat;          │I/Oポートpiopadへ出力データ
  port[ piopad ] := data; ┘data を書き出す
end;

begin
  pioinit;          ┐テストのために一つのCR
  list( $31 );      │を出力するプログラム
  list( $0D );      ┘
end.
```

これだけは **I/Oデバイスに関するデバッグ** 知っておきたい

入出力装置が関連するプログラムのデバッグは，次のような配慮が必要となります．

① 入出力装置のアドレスが正しいか？

I/Oアドレスが実装されていないアドレスを読み込むと，多くの場合ＦＦＨの値となる（データ・バスがプルアップされているため）．

② I/Oデバイスの初期化が行われているか？

初期化されているかどうかは，ステータスを読み取り，各ビットが妥当かどうかを調べます．妥当でない場合は再度初期化してみます．そのとき，初期化の手順，初期化のプログラムのチェックも行います．

初期化の前にリセットを行わなければならないものもあるので，デバイスの仕様を確認します．

③ 入出力処理では，相手側の状態によって処理が進まない場合があります．デバッグのためステータスのチェックを中断できるようにする工夫，またはステータスの状況をチェックし，相手側の状況が正常であることを確認のうえ，次に進みます．

〈図5-B〉I/Oデバイスのチェック

シリアル・インターフェース

NEXT

最初にシリアル伝送について説明します．そして，8251A/Z80 SIOの使い方とプログラミングについて説明します．

keywords

ボーレイト：直列通信における伝送速度を表す単位．１秒当たりの伝送ビット数と同義語．

マーク：直列通信時の信号のある状態．スペースは信号のない状態を示す．

モデム：変復調器．ディジタル信号とアナログ信号の変換を行い，音声（アナログ）信号でデータの伝送を行うための装置．

非同期：キャラクタの送受信間隔が任意に行える通信．各キャラクタごとにスタート，ストップを示すビットなどが付加される．

RS-232C：モデムと端末装置間のインターフェースの規格．パーソナル・コンピュータの直列非同期通信用のインターフェースを示すのにも使われる．

ENQ：enuiry．送信開始時に対する受信可の問い合わせの制御コード．

ACK：affirmative acknowledgement．問い合わせに対する肯定応答．

NAK：negative acknowledgement．受信データに誤りがあるときの否定応答．

本章では，マイコン，パーソナル・コンピュータのRS-232Cで代表される，シリアル・インターフェースについて説明します．このシリアル・インターフェースは，各メーカでの解釈の違いから，異なったメーカの製品の間での接続が，そのままでは成功しない場合が多くあります．しかし，このインターフェースの基本を理解すれば，これらの問題解決は難しいものではありません．

● シリアル・インターフェースとは接続のための信号線を節約したもの

コンピュータで扱うデータは，８ビットまたは最近では16ビットのデータを同一のタイミングで同時に処理します．したがって，コンピュータのシステムが外部の装置との間でデータの交換を行うとき，これら8，16ビットのデータを同時に受け渡しするために最低８，16本の信号線が必要となります．

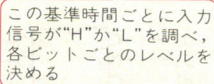
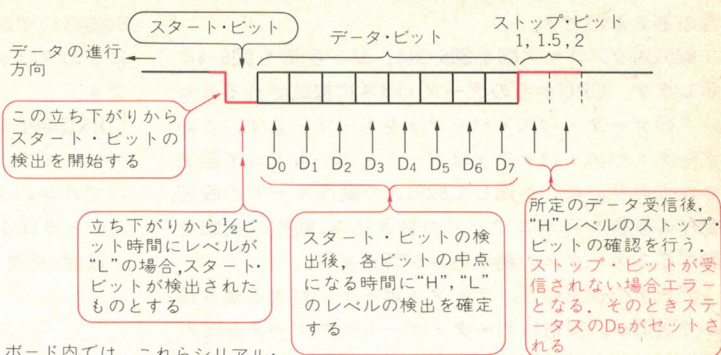
外部の装置との距離が近い場合は，信号線が多くてもそれほどコストも問題になりません．しかし，相手との距離がある場合，配線のコストが無視できなくなります．

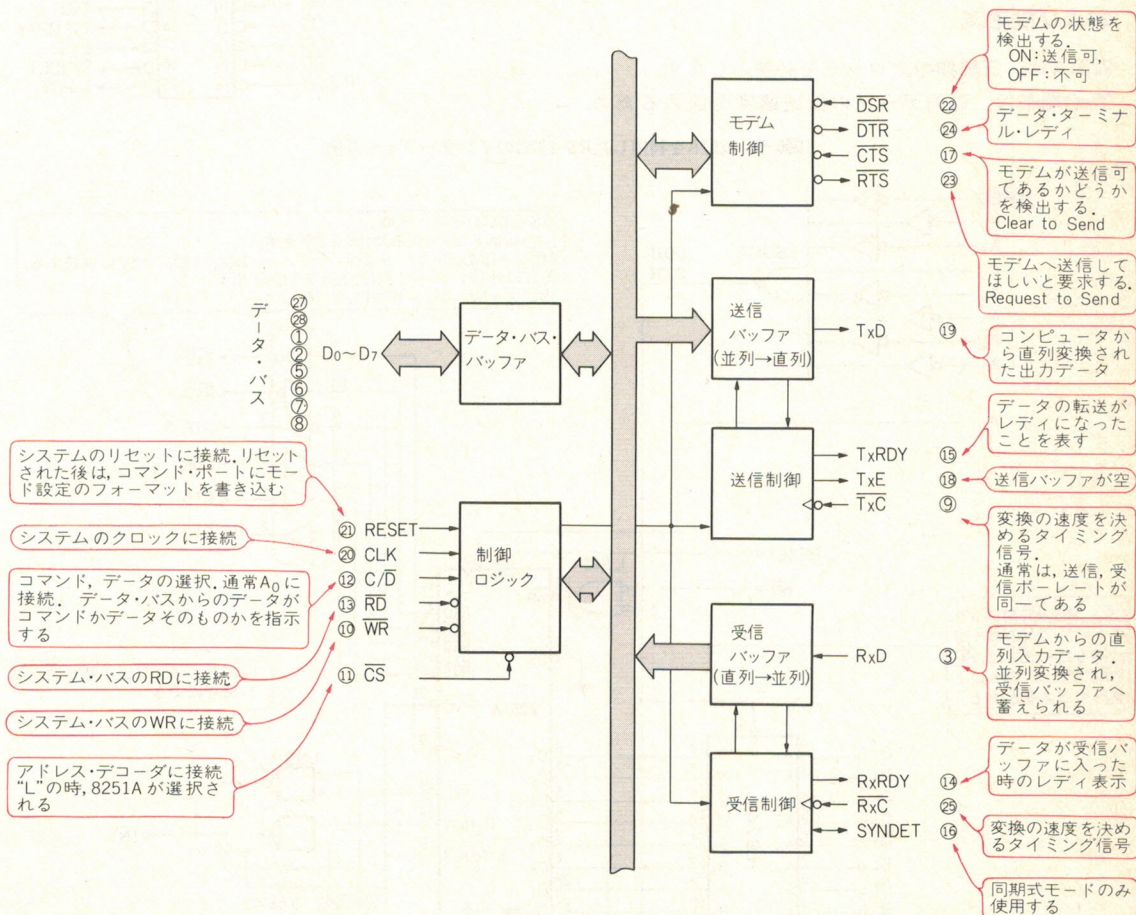
この問題解決のために，直列のデータ伝送の方法が導入されました．一方，この直列のデータ伝送は，電信システムとして長い歴史をもっていました．

● 並列のデータをどのようにして直列データに変換するか

並列データを直列データへ変換するには，図6-1に示すように１本の信号線に基準時間ごとに１ビットずつ“H”または“L”の信号を送信します．受信側では，基準時間ごとに入力信号のレベルをチェックします．所定のビット数のデータの受信後，パラレル・データに変換されます．この変換には，専用のLSIが用意されています．

この変換において，１データ分のデータのスタートをどのように検出するかが問題になります．パーソナル・コンピュータのシステムで最も一般的に利用されているのが，図6-2に示す調歩同期式と呼ばれる方法です．電信などで利用されていた方法です．

データの最初は，必ず１ビットのスタート・ビットで始まります．スタート・ビットの立ち下がり（信号がないときは“H”の状態）を基準にして，所定の伝送スピードに応じたクロックで，各ビットのON/OFFを検出していきます．データの終わりには，１から２ビットのストップ・ビットが送られます．所定の時間にストップ・ビットが検出されないとそのデータは正しくないとみなされます．

	H	L
RS-232C	+12V～+3V	−3V～−12V
RS-422	+5V～+3V	+0.4V～0V

〈図6-3〉8251Aの内部ブロック図

シリアル・インターフェース用として，最もよく使用されている8251Aの概要

現在，最もよく使用されている，シリアル通信用のデバイスは，インテル社の8251Aでしょう．この素子は，USART（Universal Synchronous / Asynchronous Receiver/Transmitter）と呼ばれ，調歩同期式以外に大型コンピュータの通信方式として利用され

ている，BSC方式などの同期式の通信も行える汎用性のあるものです．

8251Aのブロック図を図6-3に，ピン配置を図6-4に示します．CPUからのデータ・バスに接続される8ビットのデータ・バス・バッファをもっています．このデータ・バス・バッファはデータ・ポートとして設定されており，ここを通して8251Aの動作モードの設定，動作の制御を行うコマンドの書き込み，動作の状態を調べるステータスの読み込みが行えます．

データの送受信は，データ・ポートに対する書き込みで送信が，受信はデータ・ポートからデータを読み込むことでできます．

データの送信部，受信部はそれぞれ独立しています．データを受信中であっても別のデータを送信することができます．

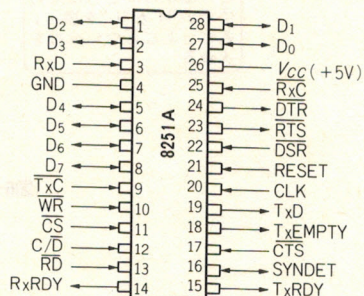
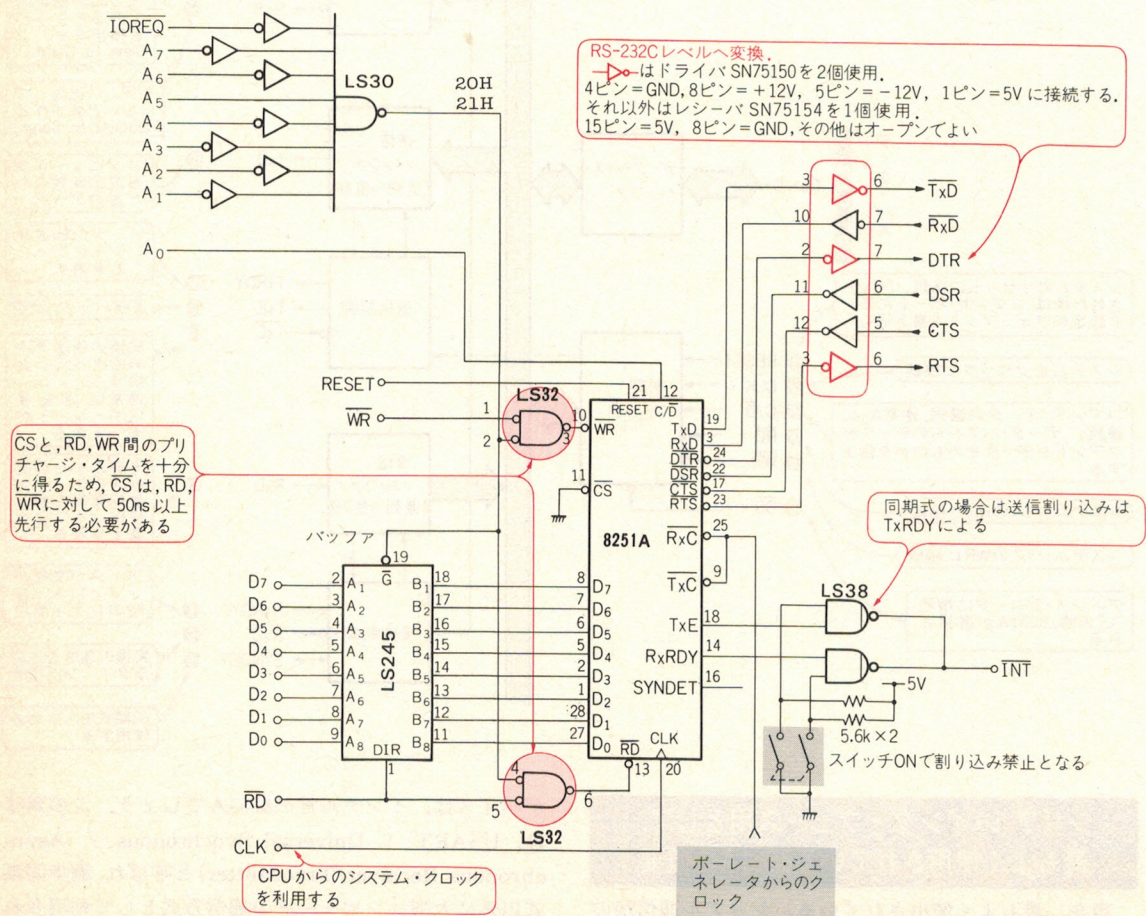
● シリアル通信の基本となるクロックの役割および設定法

8251Aは，三種類のクロックを必要とします．

TxC端子は，送信データの伝送速度を決めるためのクロックで，送信データの伝送速度に対して1，16，64倍のいずれかのクロックを加えます．同期式の場合にこのクロックは，送信データと同じ速度のもののみです．

RxC端子は，受信データのサンプリングのためのクロックで，伝送速度に対して1，16，64倍したうち，いずれかのクロックとなります．TxCと同様，同期式のときは1倍のクロックのみです．

送信，受信ともに同じ伝送速度の場合，同一のクロ

〈図6-4〉
8251Aのピン配置

〈図6-5〉 8251Aを利用したRS-232Cのインターフェース例

ックを使用することができます．

CLKは，8251Aの内部の動作のタイミング制御のために利用されます．これには3MHz以下で，TxC，RxCの30倍以上のクロックが必要です．一般にはCPUと同じか1/2に分周したシステム・クロックを用います．

これらクロックの作成方法とインターフェース回路を図6-5に示します．

● 8251Aのリード/ライト・コントロール部

8251AをCPUのシステム側から制御するために，次の五つの制御端子が用意されています．

▶C/$\overline{D}$

データ・バスに接続されているポートをコマンド処理のためのものにするか，データ処理のためのものにするかの選択を行います．

この端子が“H”のときがコマンドで，“L”のときにデータ・ポートとなります．アドレス・バスの最下位ビットのA_0を接続します．そうすると，例えばコマンドのアドレスを８１Ｈとすると８０Ｈがデータ・ポートとして連続したアドレスを指定でき，プログラム上のメリットが生じます．

▶$\overline{RD}$, $\overline{WR}$

それぞれ，システム・バスのリード/ライト信号と直接接続します．

▶$\overline{CS}$

アドレス・デコーダからのデバイスの選択信号を接続します．

▶RESET

8251Aの動作をリセットします．リセット信号は，8251Aに加えられるCLKの６サイクル分以上アクティブでなければなりません．“H”レベルです．

● 通信の相手側となるモデムとのやりとりを行う部分

コンピュータのデータ通信は，電話回線を利用して発展してきました．したがって，データ処理装置からディジタル信号として出力された信号を，電話回線で送受信可能な音声信号に変換する装置が必要となります．この変換装置をモデムと呼びます(図6-6)．パーソナル・コンピュータのカタログに載っているRS-232Cと呼ばれるインターフェースは，本来このモデムとデータ処理装置との間を接続するための規格でした．

このモデムを制御するために通常必要となる制御信号を，8251Aはもっています．

▶$\overline{DSR}$(Data Set Ready)入力

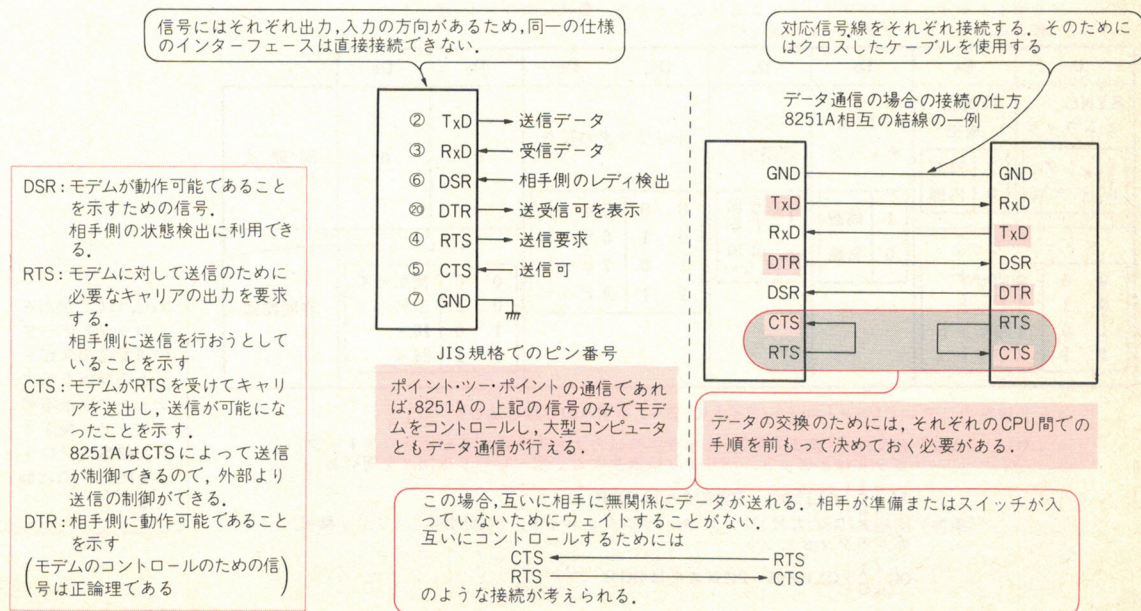
モデムが動作可能であるかどうかをこの端子で検出します．CPUからは，コマンド・ポートから読み込まれるステータスのD_7ビットで調べられます．

▶$\overline{DTR}$(Data Terminal Ready)出力

データ端末装置，8251A側が送受信可能であることを相手側に知らせます．コマンドのD_1のビットのON/OFFで，この端子は制御されます．

▶$\overline{RTS}$(Request To Send)出力

モデムに対して送信データがあることを示し，モデムが8251Aからデータを受信して相手側へデータを送信できるよう要求します．この端子の出力は，コマンドのD_5のビットのON/OFFで制御されます．

〈図6-6〉モデムのコントロール信号

▶$\overline{CTS}$(Clear To Send)入力

モデムが，8251Aからデータを受信可能かどうかを示す信号を接続します．8251Aは，この端子がアクティブ（“L”）にならないと，データを送信することができません．

外部からこの端子を制御することによって，送信の抑止が行えます．これにより，データの受け取りを確実に行うためのハンドシェイクが可能となります．

以上の説明が，モデム制御のための基本的な機能です．しかし$\overline{CTS}$以外は，8251Aの動作そのものに関係しません．したがって，ほかの目的に利用することもできますがあまりすすめられません．

● 送信部における各端子の機能

▶TxD(出力)

送信データは，この端子より出力されます．出力される条件は，コマンドでD_0のTxENのビットを“H”にし，$\overline{CTS}$の入力が“L”のときです．

この状態で，データ・ポートに送信データを書き込むとシリアルに変換され，この端子より出力されます．

▶TxRDY(出力)

データの送信が可能であることを示します．データ・バッファが空で，なおコマンドでTxENのビットが“H”で，$\overline{CTS}$が“L”のときにのみ，TxRDYが“H”のアクティブになります．したがって，この端子をCPUに対する割り込み信号として利用できます．この端子は，送信データをデータ・ポートに書き込むことでリセットされます．

この端子の状態はステータスのD_0ビットで調べることができます．

▶TxE(出力)

データの送信バッファが空になったことを示します．この信号もCPUに対する割り込み信号として使用することができます．

TxEの割り込み発生によって，CPUは送信用のバッファからデータを1バイト取り出し，送信のために8251Aのデータ・ポートに書き込みます．これにより割り込みもリセットされます．ステータスのD_2がTxE端子の状態を示しています．

▶$\overline{TxC}$(入力)

送信データに対する，タイミングを決めるクロックを加えます．

● 受信部における各端子の機能

▶RxD(入力)

受信データをこの端子で受けます．受信データの入力を禁止することはできません．したがって受信側の都合に関係なく，この端子に接続された信号が変化した場合，たとえノイズであったとしても入力データとして認識されます．

しかしノイズの場合は，フレーミング・エラー，パリティ・チェックなどのエラー・チェック処理によって検出できます．

▶RxRDY(出力)

受信データが受信バッファにセットされ，CPUからの読み込みが可能になったことを示します．

ステータスのRxRDYは，ONになることをコマンドによって禁止することができます．しかしこの端子

〈図6-7〉モード設定のコマンド

リセット端子，またはコマンドによるリセットの後，最初にコマンド・ポートに書き込まれたデータがモード設定を行う

D7	D6	D5	D4	D3	D2	D1	D0	
SYNCキャラクタ 1 シングル 0 ダブル	SYNC検出 1 外部 0 内部	パリティ・チェック 1 偶数 0 奇数	パリティ、の追加 1 追加する 0 追加しない	キャラクタの長さ 0 0 5ビット 0 1 6ビット 1 0 7ビット 1 1 8ビット		0	0	同期式
ストップ・ビット 0 0 使用せず 0 1 1ビット 1 0 1½ビット 1 1 2ビット						ボーレート 0 0 使用せず 0 1 1× 1 0 16× 1 1 64×		非同期式

RxC，TxCに加わるクロックとデータの送受信のスピードとの関係を示す．データの送受信のスピードに指定された倍数のクロックをRxC，TxCに加える

モード設定のコマンド〔コントロール・ライト($C/\overline{D}=1, \overline{WR}=0$)で書き込む〕

(例) 〔条件〕非同期式(調歩同期)，キャラクタはカナを使うので8ビット，ストップ・ビットは2ビット，パリティは無視する．ボーレートは16×を用いる．

$11\begin{Bmatrix}1\\0\end{Bmatrix}01110B$ = EEHまたはCEH

〔条件〕同期式(BSC方式)シンク・キャラクタ2バイト，内部同期，パリティ無視，キャラクタ8ビット

$00\begin{Bmatrix}1\\0\end{Bmatrix}01100B$ = 2CHまたは0CH

のRxRDYは，ソフトウェアで禁止することはできません．受信バッファを読み込むことで，この端子およびステータスのRxRDYもともにリセットされます．リセットされると“L”の状態です．コマンドのRxE＝1のとき，ステータスのD_1ビットがRxRDYの状態を示します．

▶$\overline{\text{RxC}}$(入力)

受信データのサンプリングを行うタイミングを決めるクロックの入力端子です．

▶SYNDET(入出力)

キャラクタ同期による同期通信を行う場合，同期キャラクタを受信し同期が確立したことを示す出力，または同期の検出回路を外部に設けた場合，同期の確立したことを8251Aが知るための入力となります．調歩同期式の通信の場合は，この端子は使用されません．

8251Aの動作条件を決める モードおよびコマンドの設定

8251Aは，図6-7に示すようにリセット後に設定されるモードによって多様な利用法ができます．調歩同期，キャラクタ同期，1データの大きさも5ビットから8ビット，パリティ・チェックの有無などが指定できま

〈図6-8〉コマンド命令フォーマット

D7	D6	D5	D4	D3	D2	D1	D0
EH	IR	RTS	ER	SBRK	RxE	DTR	TxEN

TxEN＝1とすることで送信可能となる．ただし，$\overline{\text{CTS}}$＝“L”とならなければデータの送り出しは起こらない

このビット状態が$\overline{\text{DTR}}$端子に現れる

受信動作そのものを禁止することはできない．ステータスとしてRxRDYが“1”になるのを禁止するだけである

送信イネーブル．1＝可，0＝不可．

データ・ターミナル・レディ．1→$\overline{\text{DTR}}$＝“L”

受信イネーブル．1＝可，0＝不可．

ブレーク・キャラクタ送出．1→TxD＝“L”

エラー・リセット．1→エラー・フラグのリセット．PE．OE．FE．

送信キャリア制御．1→$\overline{\text{RTS}}$＝“L”

内部リセット．1→初期設定へ

ハンド・モードに入る．1→SYNCキャラクタのサーチ（同期式のみ）

指定するフォーマットの実際上の動作の制御－送信／受信のイネーブル，エラー・フラグのリセット，モデムの制御などを行う．コントロール・ライト（C/$\overline{\text{D}}$＝1，$\overline{\text{WR}}$＝0）で書き込む．

送信，受信共に行う場合
EH＝0
IR＝0
RTS＝1
ER＝1
SBRK＝0
RxE＝1
DTR＝1
TxEN＝1
37Hのコマンドをコマンド・ポートへ書く

内部リセットはリセット端子によるリセットと同等である．モード設定から必要

$\overline{\text{RTS}}$＝“L”によって相手側に送信データがあることを示すのに使用されている

エラー・フラグは，このエラー・リセットによってしかクリアできない．したがって，ステータスでエラー・チェックを行っている場合，エラーが発生したら必ずこのエラー・リセットを行う必要がある

〈図6-9〉ステータスのフォーマット

このステータスはコマンド・ポートを読み込むことによって得られる

す．

モード設定によって，8251Aの動作条件が決まります．具体的な送受信を開始するためには，コマンド・ポートにコマンドの書き込みが必要です．図6-8に示すように1バイトのコマンドの各ビットに，それぞれ決められた役割が設定されています．

必要とする処理に対応するビットをON（“1”）にして，コマンド・ポートへ書き込みます．このとき，不用意に目的の処理以外の制御状態を変化させることのないように注意しなくてはなりません．図6-9にステータスのフォーマットを示します．

● 8251Aを再度初期化する場合，ダミー・コマンドの書き込みが必要

8251Aは，通常電源投入時のパワー・オン・リセット信号によってリセットされ，モード設定待ちの状態になっています．したがって次にモード，コマンドの順に処理を進めます．しかし，8251Aが現在どの状態に

〈図6-10〉
8251Aのイニシャライズ処理のフローチャート

〈リスト6-1〉 8251Aの初期化処理ルーチン

```
                        ; Z8Ø ASM exsample
                        ; Z8Ø6Ø3
                        ; i8251 init program
                        ;
ØØ21                    sioc    equ  Ø21H
ØØ4E                    mode    equ  Ø4EH
                        ;
                                .Z8Ø
ØØØØ'   AF              inisio: xor a
ØØØ1'   D3 21                   out (sioc),a
ØØØ3'   CD ØØ1B'                call dum
ØØØ6'   D3 21                   out (sioc),a
ØØØ8'   CD ØØ1B'                call dum
ØØØB'   D3 21                   out (sioc),a
ØØØD'   CD ØØ1B'                call dum
ØØ1Ø'   3E 4Ø                   ld a,4ØH
ØØ12'   D3 21                   out (sioc),a
ØØ14'   CD ØØ1B'                call dum
ØØ17'   3E 4E                   ld a,mode
ØØ19'   D3 21                   out (sioc),a
                        ;
ØØ1B'   C9              dum:    ret
                        ;
                                end
```


あるのか不明な場合があります．

そのようなときには，コマンドの内部リセットの機能により現状をリセットして，再度モード設定より行い，目的の状態にすることができます．しかし，内部リセット・コマンドを書き込んだとき，モード設定待ちまたはBSCモードのSYNCキャラクタのロード待ちの状態では，8251Aはコマンドとしては処理しません．

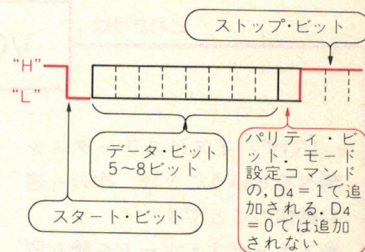
したがって，コマンドによる内部リセットを行う場合，コマンドに先立って3回，00Hなどのリセット・ビットがOFFになっているコマンドを書き込みます．

これにより，それ以前にどのような状態であろうと，確実にコマンド受け付け可の状態になります(図6-10)．

これらのことを考慮したプログラムの例を，リスト6-1として示します．

● パリティ・イネーブルで1ビットのパリティ・ビットを付加する

モード設定で，パリティ・ビットを付加して送受信時のエラー検出を行うことができます．この場合図6-11に示すように，所定のデータの後に1ビットのパリティ・ビットが追加されてデータが送信されます．このパリティ・ビットは，パリティ・エラーの有無のチェックが行われた後，受信側で自動的に取り去られます．データ・バスを介して，CPUは送信側が8251Aに渡したパリティ・ビットなしの元のデータを受け取り

〈図6-11〉
パリティ・ビットの扱いについて

ます．

● 調歩同期式の送信はポツポツ送ることもできる

調歩同期式の送信は図6-11に示されているように，データの終了を示すストップ・ビットが"H"レベルで，そのまま次のデータが送信されないとしても，受信側は次のデータのスタート・ビットの"L"への立ち下がりまで待っているだけです．これは，各データのフレームごとにデータの範囲を示すビットをもっているため，1データずつ単独で送信したとしても問題は生じません．

しかし，このために実際に必要とするデータの2割から4割も余分なビットを，送受信していることになります．キーボードとコンソールの通信のように回線

のコストが低く，伝送効率がそんなに問題にならない場合，ハードウェア，ソフトウェアともに簡単ですみますので調歩同期式は便利な方法です．

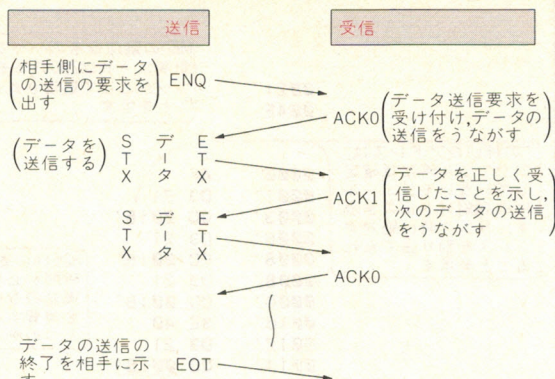
● 高速な受信処理は割り込み処理が有利になる

入力したデータにはなんらかの処理を施すのが普通です．その場合，高速での連続したデータ受信では，データのドロップ・アウトが生じる可能性があります．

RS-232Cのインターフェースでも，パラレルI/Oのようにハードウェアでのハンドシェイクも実現できます．しかしそのためには，往復の信号線以外に制御のための線が必要になります．信号線の数が少なくてすむという，シリアル伝送の意味がなくなってしまいます．

データの送受信を1本の信号線で実現することもできます．アース線が1本必要なために，最低2本のラインが必要となります．

したがって一般的には，シリアル伝送では，図6-12に示すようなデータの受け渡しの手順が，プロトコルと呼ばれるものとして厳密に決められています．この場合でも最小限，送受信の最小単位である1レコードの受信は，ドロップ・アウトなしに確実に受信できなければなりません．

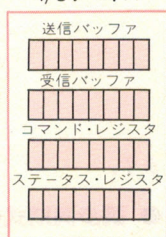
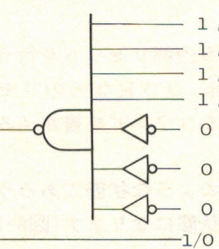
〈図6-12〉伝送プロトコルの概要

このようにデータの受け渡しの手順が決められている．JIS-C6362でより詳細に，またいろいろな局面についても具体的に決められている．
ENQ, STX, ETX, ACK0, ACK1, EOTは，伝送制御用のコードとして決められた1バイトのコード．

そのためには，データの受信および受信バッファへの書き込みを，割り込み処理で行うことで容易に実現できます．割り込み処理によるデータ受信の具体的な例は第8章で説明します．

これだけは　知っておきたい

I/Oデバイスのステータスを知ること

I/Oデバイスのステータスを調べるということは，次の一連の動作のことです．

◈ステータス・ポートを読み取る
▶ポートのアドレスを出力
▶I/Oデバイスからステータス・データをデータ・バスへ出力させる．
▶CPUのレジスタがそのデータを保存する
◈所定のビットのON/OFFを調べる
▶ビットのON/OFFに応じて，ZフラグがON/OFFする命令を実行
▶ZフラグのON/OFFに応じて，ジャンプ先の異なる分岐命令で，ステータスの状況に応じた処理に移る

〈図6-A〉I/Oデバイスの内部レジスタを選択するようす

● 送信終了時の送信停止のためのコマンドは注意しないと送信途中のデータを失う

送信を終了した後に，TxENをOFFにしたコマンドを書き込み，送信機能を停止する場合があります．そのとき，コマンドの書き込みタイミングによっては，送信データが失われます．これは，送信バッファ中に送信データがあるときにD_0ビットを0にした送信ディセーブルのコマンドを書き込むと，バッファ中のデータが消失するためです．

これを防ぐには，データの送信終了後の送信停止のコマンドをTxEのフラグを検出し，送信バッファ中にデータが残っていないことを確認した後で行います．TxEのチェックは，ステータスのD_2ビットが1になっていたなら，バッファにデータは残っていないとして処理できます．

また，割り込みを使用しない送信の場合は，とくに送信をディセーブルしなくても問題は生じません．

しかし，データ送信のタイミングを割り込みによって検出している場合，データ送信後は不要な割り込みが生じないように，8251Aの送信機能を停止しなければなりません．その場合上記の注意が必要となります．

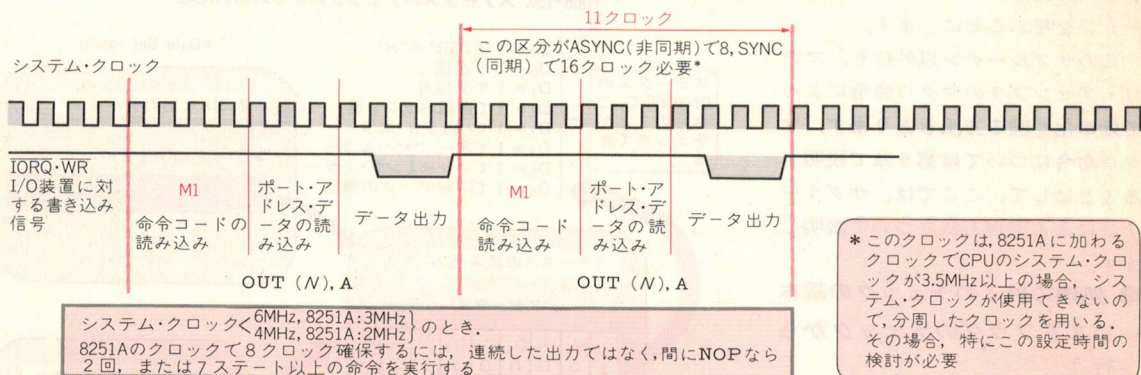
送信時には，同期通信以外では割り込み処理はあまり行われていません．一般的な使用法ではあまり問題にならないかもしれません．

● 8251Aの実行速度

CPUと8251Aのクロックが異なる場合は，注意が必要です．8251Aの内部処理の動作は，8251AのCLK端子に加えられたクロックに同期して行われます．CPUの処理速度が速い場合，この8251Aのクロックはシステム・クロックを分周したものを使用することになります．そのような場合，CPUの命令の実行速度にくらべ8251Aの内部の処理速度が遅れるので，それぞれの実行速度を確認しておく必要があります(図6-13)．

● 処理の終了の時間待ちには無効果な命令を実行する

I/Oデバイスなどの実行の処理の終了を待つ場合，時間つぶしのためのプログラムとして，現在の処理に効果のない命令を実行します．それにより外部のデバイスの処理の終了を待つことができます．この方法に用いる命令は，外部の実行中の処理のスタートから終了までに見合った時間で，その命令の実行が終了することが保証されている必要があります．

〈図6-13〉 コマンド書き込み時のコマンドの設定時間の検討

〈図6-14〉 遅延処理のための命令実行

(a) NOP命令を使う方法

命令	遅延クロック数	処理
NOP	4クロック	無効果．何の処理も行わない

機械語のコードは00Hとなる．必要な遅延時間だけ続ける

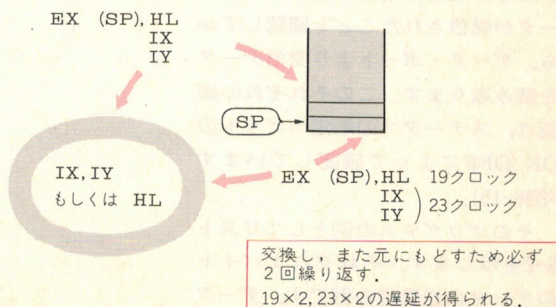
遅延時間(μs)＝NOPの命令の数 × 命令の実行クロック数 / システム・クロックの周波数(MHz)

(例)
NOP
NOP
NOP
NOP
NOP
NOP

▶4MHz $6 \times 4/4 = 6\mu s$

▶6MHz $6 \times 4/6 = 4\mu s$

(b) スタックを使う方法

外部からの割り込みが禁止されている場合，スタックを利用した遅延が利用できる．

その時間も，一般的な外部デバイスの処理時間と命令の処理クロックの数から，(システム・クロックの周期)×(4～30)くらいの時間が妥当なところです．I/Oなどの外部デバイスの処理時間は，メーカや素子の作り方が変更され(C-MOS化)，変わる場合もあります．

また，システム・クロックが変われば，処理時間も変わってしまいます．システムのバージョンアップでクロックが速くなったらプログラムが動かなくなった，というようなことのないようにしてください．

図6-14に，遅延のための命令と遅延時間を示しておきます．

処理はサブルーチン，またはマクロ命令化することでプログラムの生産性をあげられる

初期化がすんだら，その後は具体的なデータの送受信の処理となります．データの送受信には多くの方式があります．ここでは，その中の半二重と呼ばれる送信と受信を交互に行う方法の場合の，送信ルーチンと受信ルーチンを考えてみます．

このとき，それぞれの処理ルーチンは，プログラムの複数の場所で使用することになります．そのたびに同じコーディングを行うのでは煩わしすぎます．そこで，このそれぞれのルーチンをサブルーチンとして作成し，必要とする場所でこの処理ルーチンを呼ぶことにします．

このサブルーチン以外にも，マクロ・アセンブラのマクロ命令によって効率化を図る方法もあります．マクロ命令については第9章で説明することにして，ここでは，サブルーチンによる実現方法について説明します．

● 処理の終了のチェックの基本はまずフラグのチェックから行う

8251Aのデータの送信は，デバイスが送信可であることを確認してからデータ・ポートに送信データを書き込み，受信の場合は相手からのデータが受信されたことを確認してから，データ・ポートより受信データを読み取ります．このそれぞれの確認は，ステータスの所定のビットのON/OFFによって確認しています(図6-15)．

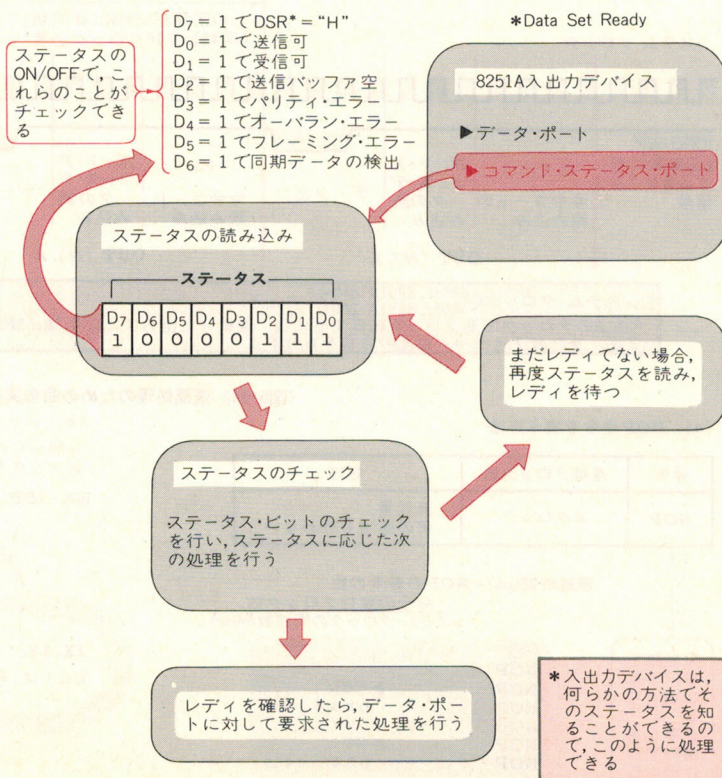
そのプログラムの例としてリスト6-2を示します．それぞれ1バイトのデータの送受信の例です．データの受け渡しについては，入力データはAレジスタ，出力データはCレジスタを経由して行っています．

連続して多数のデータの入出力を行う場合，所定のデータ・エリアをデータの受け渡しのために用意する場合があります．この入出力のためのデータ領域をバッファと呼び，割り込み操作のときに不可欠な技術です．詳しくは第8章で説明します．

● プログラムの作成でシステムが希望どおりの動作をしない場合の対策が不可欠

リスト6-2で示したプログラムは，相手からのデータがこないとき，なんらかの不都合で相手側が受信状態でない場合など，ただステータスの読み取りチェックを繰り返すだけで，このルーチンから抜け出すことができません．処理の再開のためには，CPUのリセットをしなくてはならないなどということになります(図6-16)．

これを避けるために，フラグをチェックし続ける時間に制限を設ける必要があります．この時間制限のためには，マルチ・ジョブのモニタなどでは，ハードウェアによるタイマを用いる方法があります．市販のパソコンなどの通信用のソフトなどでも，ハードウェアのタイマによる時間管理が一般的です．しかし，普通はそれほどの時間管理が要求されているわけでなく，

〈図6-15〉ステータスのチェックによる入出力処理

〈リスト6-2〉 8251Aの送信と受信のサブルーチン

```
                    ;    Z8Ø ASM exsample
                    ;    Z8Ø6Ø4
                    ;    i8251
                    ;
ØØ21                sioc     equ   Ø21H      ;コマンド・ポート・アドレス
ØØ2Ø                siod     equ   Ø2ØH      ;データ・ポート・アドレス
                    ;
                             .Z8Ø
                    ;送信ルーチン
ØØØØ'   DB 21       datatx:  in a,(sioc)
ØØØ2'   CB 47                bit Ø,a
ØØØ4'   28 FA                jr z,datatx
                    ;
ØØØ6'   79                   ld a,c
ØØØ7'   D3 2Ø                out (siod),a
ØØØ9'   C9                   ret
                    ;受信ルーチン
ØØØA'   DB 21       datarx:  in a,(sioc)
ØØØC'   CB 4F                bit 1,a
ØØØE'   28 FA                jr z,datarx
                    ;
ØØ1Ø'   DB 2Ø                in a,(siod)
ØØ12'   C9                   ret
                    ;
                             end
```


〈リスト6-3〉 無限ループを避けた送受信のサブルーチン

所定の時間 入出力がなければ，その旨をセットし，次に進む入出力ルーチンの例

```
                    ;  Z8Ø  ASM exsample
                    ;  Z8Ø6Ø5
                    ;
ØØ21                sioc      equ Ø21H
ØØ2Ø                siod      equ Ø2ØH
                    ;
ØBB8                lpcnt     equ Ø3ØØØ
                    ;
                              .Z8Ø
ØØØØ'   11 ØBB8     datatl:   ld de,lpcnt ;繰り返しの回数をセットする
ØØØ3'   DB 21       lptd:     in a,(sioc) ;ステータスのチェック
ØØØ5'   CB 47                 bit Ø,a
ØØØ7'   2Ø Ø6                 jr nz,td
ØØØ9'   1B                    dec de
ØØØA'   7A                    ld a,d
ØØØB'   B3                    or e
ØØØC'   C8                    ret z
ØØØD'   18 F4                 jr lptd ;ゼロならば所定の回数になったから不成功としてそのままもどる
                    ;
ØØØF'   79          td:       ld a,c
ØØ1Ø'   D3 2Ø                 out (siod),a
ØØ12'   C9                    ret
                    ;
ØØ13'   11 ØBB8     datarl:   ld de,lpcnt
ØØ16'   DB 21       lprd:     in a,(sioc)
ØØ18'   CB 4F                 bit 1,a
ØØ1A'   2Ø Ø6                 jr nz,rd
ØØ1C'   1B                    dec de
ØØ1D'   7A                    ld a,d
ØØ1E'   B3                    or e
ØØ1F'   C8                    ret z
ØØ2Ø'   18 F4                 jr lprd
                    ;
ØØ22'   DB 2Ø       rd:       in a,(siod)
ØØ24'   C9                    ret
                    ;
                              end
```

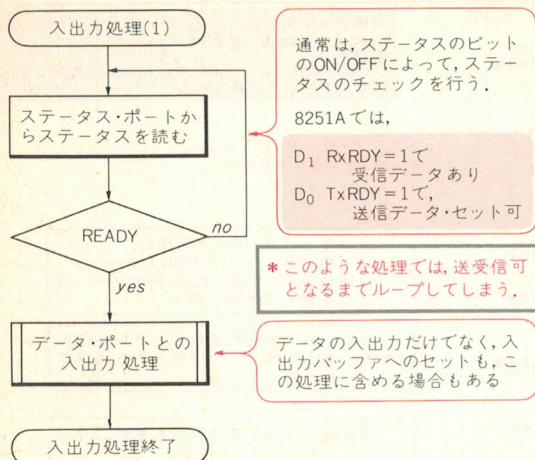
- ここに示した回数分，送受信のチェックを行い，成功しなければ次に進む（lpcnt equ Ø3ØØØ）
- 送信可であればこのループを抜け，送信作業に移る（jr nz,td）
- DEレジスタを−1して，ゼロになるかのチェックを行う（dec de ～ ret z）
- Zフラグが OFFになってこの場所へくる．送信データをデータ・ポートへ出力して，メイン・ルーチンへもどる（td:）
- 受信データがないとき，ZフラグがON（bit 1,a）
- 所定の回数繰り返したときにZフラグがON（or e）
- 不成功のときにZフラグONでもどる（ret z）
- 16ビットのペア・レジスタのゼロのチェックは，ペア・レジスタの一方をAレジスタに転送し，Aレジスタと残りのレジスタの論理和を調べる．結果がゼロならば，ペア・レジスタはゼロである
- Aレジスタにデータを得てZフラグOFFでもどる（rd:）

〈図6-16〉ステータス・チェックによる入出力処理フローチャート

〈図6-17〉処理時間の計算

lptd: in a,(SIOC)	11	11
bit 0,a	8	8
jr nz,td	7	7
dec de	6	6
ld a,d	4	4
or e	4	4
ret z	5	11
jr lptd	12	
合　　計	57ステート	

$$処理時間=ステート数\times\frac{1}{システム・クロック周波数}$$
$$=57\times(1/4\,\mathrm{MHz})$$
$$=57\times250\,\mathrm{ns}$$
$$=14250\,\mathrm{ns}=14.25\,\mu\mathrm{s}$$

全体の処理時間＝処理時間×繰り返しの回数

ループの回数を管理して制限することで目的を達成できます．

その具体的なプログラムを**リスト6-3**に示します．ループ・カウンタにセットする値を変えることで，待ち時間を制御することができます．この場合の待ち時間の計算方法は，**図6-17**に示すようになります．この方法では，CPUの処理速度(システム・クロック)が変わったときには，待ち時間が変わることに留意しておいてください．

● 期待どおりの動作ができなかったときそれを表示する必要がある

サブルーチンに渡されたそれぞれの処理の結果が正しく行われたか，なんらかのトラブルが生じたかを元のプログラムに知らせる必要があります．データが送信できたのか，受信できたのか，または相手からの応答がなくタイムアウトでもどってきたのか，識別できる方法を考えなければなりません．

この識別を行うものとしてフラグの利用を考えます．このフラグとしてZ80のフラグ・レジスタのそれぞれのフラグ・ビットを利用するか，ほかのレジスタ，またはメモリ中のデータを利用することもできます．

Zフラグの ON/OFFによる識別がわかりやすく，よく使われています．**リスト6-3**の例では，入出力いずれのステータスも，それぞれのビットがONのとき送受信が可能となり，入出力操作が行われます．しかし相手側が準備できていないなどの理由で不調なときは，ステータス・ビットがOFFとなっており，ビットのOFFを示すZフラグがONとなっています．

〈図6-18〉無限ループに陥らない入出力処理のフローチャート

そのうえ，繰り返し回数の限界を示す設定値のカウント・ダウンが終了すると，カウント値がゼロとなりZフラグがONとなります．

したがって，**リスト6-3**の例では，この入出力のサブルーチンを終えてきたとき，ZフラグがONであれば入出力が不成功であり，ZフラグがOFFであれば入出力操作が正常に終了したことが示されます(**図6-18**)．

カウンタ/タイマの使い方

NEXT

8253とZ80 CTCのカウンタ/タイマ機能の，基本的な使い方は同じです．ボーレート・ジェネレータを例にプログラミングを説明します．

keywords

ダウン・カウンタ：カウンタに設定された値を入力クロックごとに－1し，0になった時点で出力の反転，割り込みの発生などで外部に知らせることができるカウンタ．

PIT：Programmable Interval Timer．インテル社のタイマ用のデバイス．8253．

CTC：Counter Timer Circuit．Z80ファミリのタイマ用のデバイス．

プリスケーラ：高い周波数のクロックを許容範囲にするための分周器．CTCの場合，システム・クロックを分周する．

トリガ：引金．カウンタの動作開始のきっかけを与えるもの．外部のパルス，コマンドなどがトリガとなる．

コマンド・ポート：コマンドの入力のためのデバイスに用意された入力ポート．

カウンタ・ラッチ：16ビットのカウンタなどのように複数回の読み込みが必要な場合，カウンタの値を正しく読み出すためラッチ・レジスタに移してから読み出す．

各マイクロコンピュータ・ファミリには，それぞれカウンタ/タイマ用のLSIが用意されています．このことは，マイコンのシステムにおいてもカウンタやタイマが基本的に必要な機能で，各応用でよく利用されるからです．Z80の応用システムでは，インテル社の8253とザイログ社のZ80 CTCがよく使用されます．

今回は，具体的なプログラムの例として最初に，8253でシリアル通信に必要なボーレート・ジェネレータについて考えてみます．その後，Z80 CTCの仕様について説明します．

● プログラマブルな周辺デバイスは使用する前に仕様を決める初期設定が必要

ここまでに説明した8255Aや8251Aもそうですが，プログラマブルな周辺用のデバイス(LSI)は，その初期設定によって多様な利用方法が選定できるようになっています．そのため，ハードウェアを変更することなくシステムに多様な機能を盛り込むことができます．

この周辺用のデバイスは，CPUの動作と独立に，指定された処理を行う機能をもっています．そのためCPUは，デバイスに対して，処理コマンドやデータなどを設定して，デバイスの処理が終わるのを待つだけですみます．このことは，CPUおよびソフトウェアの負荷を少なくすることにもなります．

そのうえ，初期設定によってデバイスを目的に応じて変身させることができるので，自由度が高い柔軟なシステムを構築できます．

カウンタ/タイマによるボーレート・ジェネレータはプログラムだけで通信の速度を変えられる

シリアル通信の処理では，その通信速度を規定する送受信データのタイミングをとるために，伝送系のクロックの速度が厳密に決められています．一般的には表7-1に示すような速度となっています．そして，これら通信処理用のデバイスでは，その速度に応じた送受信用のクロックを必要とします．

今回ここでは，その通信用のデバイスに供給するク

〈表7-1〉 一般的なシリアル通信の伝送速度

伝送速度* (bit/sec)	16倍クロック** (Hz)
110	1760
300	4800
600	9600
1200	19200
2400	38400
4800	76800
9600	153600
19200	307200

* シリアル通信の伝送速度 1秒間に送信されるビット数を単位として表すのが一般的(bit/sec)

** 非同期通信用のLSIは多くの場合，伝送速度の16倍の周波数のクロックでタイミングをとる

ロックを，カウンタ/タイマ用のデバイスを用いて実現することにします．

このカウンタ/タイマは，図7-1に示す仕様のうち，基準となるクロックを所定の回数分周する機能を用いて，必要とする周波数のクロックを得ています．

この入力は，不定期に発生するパルス信号を検出することもできます．この入力パルスを計数することで，カウンタとしての機能を実現することもできます．これらの実現される機能は，すべて初期設定時のパラメータの値によって決まるのが普通です．具体的な回路を図7-2に示します．

● 8253の仕様は専用のコマンド・ポートに仕様を示すコマンド・ワードを書き込む

このカウンタ/タイマを使用する前には，必ずコマンド・ポートに使用するチャネルの使用条件を決めるためのパラメータを書き込みます．

〈図7-1〉プログラマブルなカウンタ/タイマ(8253)の概念図

8253を想定している．ほかのタイマ用素子の仕様は異なるが，基本的な概念は大きく変わらない．具体的な素子の仕様は，マイコン周辺LSI規格表などを参考のこと

〈図7-2〉8253の使用例

アドレス・デコーダは，アドレス信号線のみで構成する

Z80 CPUの$\overline{RD}$,$\overline{WR}$信号と，$\overline{IORQ}$のそれぞれかけあわせた，I/O用のリード/ライト信号

1番ピンのDIRは，A→Bへデータが流れるか，B→Aへデータが流れるかという方向を決める信号

システム・バスの負荷が大きく，8253のドライブ能力(I_{OL}:2.2mA)で不足する場合，バッファを設ける．通常のシングル・ボードのシステムでは，ほとんど必要としない

選択されるレジスタ名	端子名					処理
	$\overline{CS}$	A_0	A_1	$\overline{RD}$	$\overline{WR}$	
0	L	L	L	H	L	カウンタ0にカウント値をセット
0	L	L	L		H	カウンタ0のカウント結果を読む
1	L	H	L	H	L	カウンタ1にカウンタ値をセット
1	L	H	L	L	H	カウンタ1のカウント結果を読む
2	L	L	H	H	L	カウント2にカウント値をセット
2	L	L	H	L	H	カウント2のカウント結果を読む
	L	H	H			不可
コントロール	L	H	H	H	L	各カウントへのコマンドの設定

このコマンド・ワードの仕様を表7-2に示します．このカウンタ/タイマは，この表中MODEのところに示すような六つの使用方法があります．今回はその中で，モード３のパルス・ジェネレータの機能を用います．

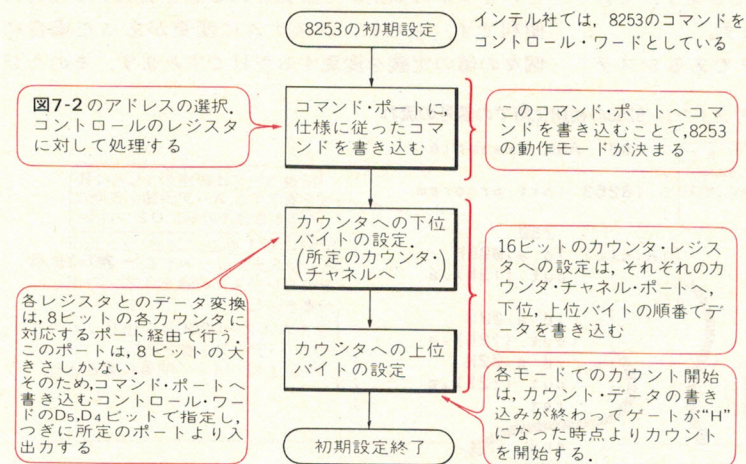
〈図7-3〉 8253の初期設定法

その場合，コマンド設定の手順は，図7-3に示すような簡単なものとなります．

コマンドを設定し必要とするカウント値を書き込むと，出力端子から所定の周波数のクロックが出力され続けます．クロックの周波数の変更は再度カウント・データを設定しなおすことでできます．そのため，このカウント・データの変更をソフトで実現することにより，通信速度を変更することが可能となります．

カウント・データの再セットの場合でも，カウント・データのセットの前にはコマンドの必要なことを忘れないでください．

● パラメータ，ポート・アドレスを定数として与える初期設定プログラム

このカウンタ/タイマの初期設定プログラムをそのまま記述すると，

〈表7-2〉 8253のコントロール・ワード

■コントロール・ワード

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC_1	SC_0	RL_1	RL_0	M_2	M_1	M_0	BCD

(MSB)　　(LSB)

▶各カウンタの動作モードを設定するために，コントロール・ワード・レジスタを選択し，コントロール・ワードを書き込み，モード設定した後，カウント・レジスタにカウントを送る必要がある

● SC(セレクト・カウンタ)

SC_1	SC_0	意　味
0	0	カウンタ０選択
0	1	カウンタ１選択
1	0	カウンタ２選択
1	1	無効

▶SC_1, SC_0により選ばれるカウンタのみが動作設定される

● RL(リード/ロード)

RL_1	RL_0	意　味
0	0	カウンタ・ラッチ・オペレーション
0	1	下位８ビットのみの読み出し/書き込み
1	0	上位８ビットのみの読み出し/書き込み
1	1	下位，上位で16ビット・レジスタ全体の読み出し/書き込み

▶RL_1, RL_2によりカウント・レジスタとのデータのやりとりの仕様が指定される

▶カウンタ・ラッチ・オペレーションは，カウント動作をみだすことなくカウンタを読み取る

▶下位または，上位の８ビットのみのレジスタの読み書きの場合，ほかの８ビットは０とみなされる

▶16ビット・レジスタ全体の読み書きの場合は，必ず下位，上位の順番で８ビットずつ処理する

● MODE(モード設定)

M_2	M_1	M_0	モード名	内　容
0	0	0	モード０	カウント完了割り込み
0	0	1	モード１	プログラマブル・ワンショット
×	1	0	モード２	繰り返し波形を発生
×	1	1	モード３	方形波の繰り返し発生
1	0	0	モード４	ソフトウェア・トリガ・ストローブ
1	0	1	モード５	ハードウェア・トリガ・ストローブ

×＝無効果（いずれでもよい）

▶M_2, M_1, M_0によりカウンタの動作モードが設定される

● BCD

BCD	意　味
0	16ビット・バイナリ・カウンタ
1	BCD(４桁)カウンタ

BCD＝“0”：16ビット・バイナリ・カウンタ　(0000H～FFFFH)

BCD＝“1”：BCDカウンタ　(0000D～9999D)

BCDにより，16ビット・バイナリ・カウンタとして動作するか，BCDカウンタとして動作するかが設定される

このコントロール・ワードは，A_1＝１, A_0＝１で選択されるコントロール・ワード・レジスタに書き込み，各チャネルの動作モードの設定を行う．
各チャネルのデータのリード/ライトは，A_1, A_0で選択されるチャネルのレジスタに対して行う

リスト7-1のようになります．

ここでは，前もってモード設定のパラメータを，定数の各ビットとして合成しておかなければなりません．リスト中ではＢ６Ｈとなっています．これは表7-2の各ビットの役割を決めて16進表現としたものです．その処理をアセンブラに任せることもできます．その場合のリストをリスト7-2として示します．

このプログラムでは，入力クロックであるシステム・クロックから，所定のクロックへ分周するための，分周比の計算などもアセンブラが行っています．

アセンブラでは，オペランド，定数の定義などを行う場合に，式を指定することができます．この機能が利用できると，システム・クロックの周波数，分周比などいくつかの要素から計算される値を使用する場合に便利です．さらに，システムに変更があった場合に個々の値の定義を変更するだけですみます．そのたび

〈リスト7-1〉 8253の初期化時での信号の流れ

〈リスト7-2〉 8253のボーレート・ジェネレータとしての初期化

に，電卓で計算する必要がありません．そのために，計算ミス，変更漏れなどによるエラーを避けることができます．

みかけ上，ソース・リストは大きくなっています．しかしこれらの処理は，アセンブラがソース・プログラムをアセンブルするときの処理のためのもので，実行時のプログラムに対してはなんら影響を与えていません．おおいに利用すべきです．

● 定数の定義，算術演算子，論理演算子などを利用することでコーディングの効率をあげられる（図7-4）

複数の定数の関係がプログラムのコーディング時に決まっていて，それらが演算によって得られる場合，前記のリスト7-2のようにアセンブラに演算させます．

このアセンブラに処理させる演算は，アセンブル時にアセンブラが定数，ラベルのアドレスなどを計算するために行うもので，機械のコードとしてアセンブルされたものには結果の値しかありません．そのため，プログラムで計算するのと異なって，実行時に計算に時間がかかるようなことはありません．

このようにプログラムによる設定だけで，カウンタ/タイマは多様な仕様をもつことができます．

Z80 CTC

Z80 CTCは，Z80ファミリのカウンタ/タイマ用の専用デバイスです．28ピンのDIPパッケージにおさめられています．

このデバイスは，次のような特徴をもっています．

▶Z80の各モードに対応した割り込み処理機能をもっている

▶Z80のデイジィ・チェーンによる割り込み優先順位処理の機能を内蔵している

▶外部からのクロックまたはトリガ入力端子をもっている．

▶システム・クロックをクロック源とすることができる．その場合プリスケーラによって1/16または1/256に分周される

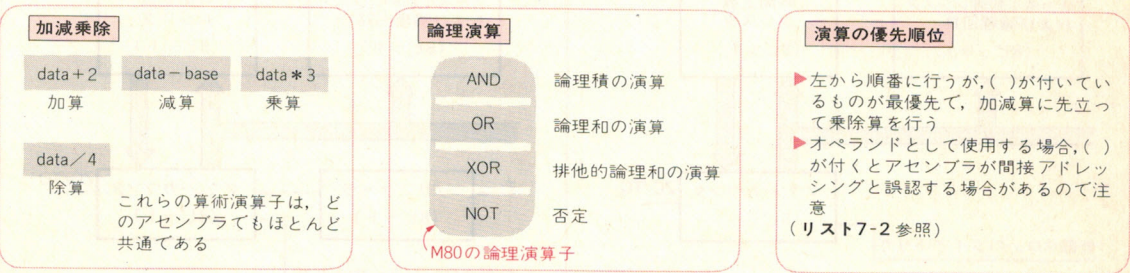
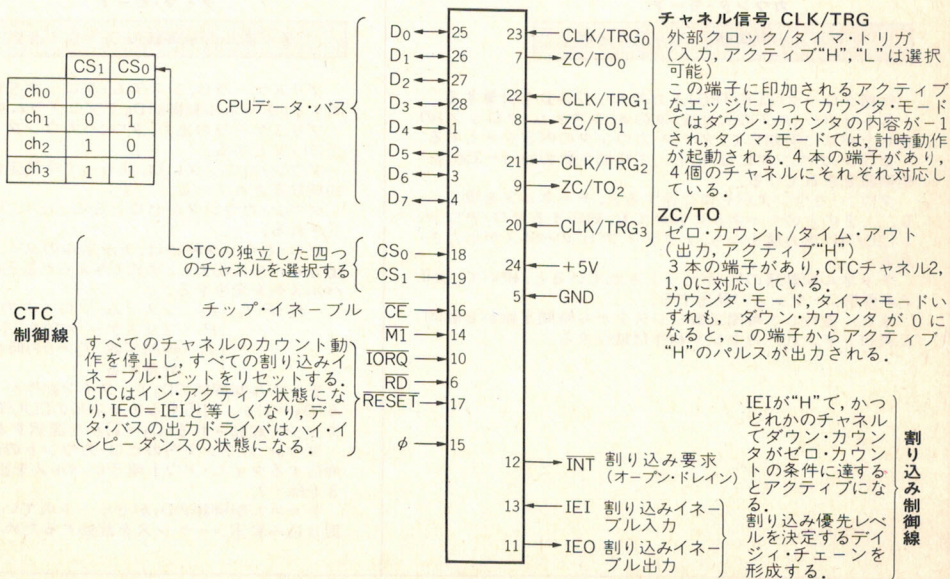
▶タイマ・モード，カウンタ・モードの二つのモードを

〈図7-4〉アセンブラが行う演算処理

演算の優先順位

▶左から順番に行うが，( )が付いているものが最優先で，加減算に先立って乗除算を行う

▶オペランドとして使用する場合，( )が付くとアセンブラが間接アドレッシングと誤認する場合があるので注意

（リスト7-2参照）

〈図7-5〉Z80 CTCの端子配置と内部ブロック図

もっている．これらのモードおよび動作状態はプログラマブルに設定できる．

● ピン配置および内部ブロック

ピン配置図およびブロック図を図7-5，図7-6に示します．各ピンの働きについて，次に説明します．

D_0～D_7：データ・バスで，CPUのデータ・バスに接続し，各チャネルへのコマンド，カウンタ，タイマの設定値の書き込みに利用される．各チャネルのカウント値を読み出すこともできる．

CS_0，CS_1：内部の4チャネルの各タイマ，カウンタ・レジスタの選択を行う．

CLK/TRG：外部からのクロックなどの入力端子．

ZC/TO：カウント・レジスタがカウント・ダウンの結果0となると，この端子が"H"となる．CLK/TRGからの入力クロックを分周した結果が得られる．

外部ピン数の制限からチャネル3については，このZC/TOの端子をもっていません．そのため，チャネル3は外部出力を必要としないアプリケーションに使用します．

● Z80 CTCの動作モード

CTCは図7-7に示すように，カウンタ・モード/タイマ・モードの二つのモードをもっています．

これらのモードの設定は，各チャネルごとにチャネ

〈図7-6〉Z80 CTCブロック図

〈図7-7〉Z80 CTCの動作モード

カウンタ・モード

チャネルの制御語の $D_6=1$ で選択

外部クロック（CLK/TRG）入力のエッジの数を計算する．

各トリガ・パルスの入力後，次のφ（システム・クロック）の立ち上がりに同期して，ダウン・カウンタの内容を－1する．

ダウン・カウンタには，前もって時間定数の値（1～256）が初期設定されている．

ダウン・カウンタがゼロに達すると，チャネル3を除く，0，1，2のどのチャネルにおいても，対応するゼロ・カウント（ZC/TO）出力端子から，アクティブ"H"のパルスが出力される．

チャネル制御語の D_7 がセットされていると，続いて，割り込み要求シーケンスが発生する．

また，並行して時間定数レジスタから時間定数が自動的に再ロードされ，カウント動作は続行する．

タイマ・モード

チャネルの制御語の $D_6=0$ で選択

プリスケーラは，システム・クロックを16または256で分周する（チャネル制御語 D_5 でプログラムする）．

プリスケーラの出力，ダウン・カウンタを－1するためのクロックとする．

ダウン・カウンタには，前もって時間定数の値（1～256）が初期設定されている．

ダウン・カウンタがゼロとなるごとに，時間定数が再ロードされる．

ゼロ・カウンタ時には，チャネルのタイム・アウト（ZC/TO）出力もパルス出力し，次式で与えられる正確な周期の一様なパルス列を発生する．

$t_c \cdot P \cdot T_c$
- t_c：システム・クロックの周期
- P：プリスケーラの値．16か256
- T_c：時間定数．$T_c=0$ の時は256として計算する

チャネル制御語の D_3 で，タイマ動作が自動的に起動される（$D_3=0$）か，対応するチャネルのCLK/TRG入力のトリガ・エッジで起動される（$D_3=1$）かを選択する．

ダウン・カウンタ内のゼロ・カウントの条件成立により，対応するタイム・アウト端子にパルスを出力する（チャネル3を除く）．

チャネル制御語の D_7 がセットされていると，この条件は，割り込み要求シーケンスを起動するために使用される．

ル制御語を設定することで行います．Z80 CTCは各チャネルごとに必要とする制御語およびカウント値を設定し，8253のような制御語専用のポートはもっていません(図7-8)．

CS_0, CS_1で選択される，四つのチャネルを独立に動作させることができます．

割り込みベクトルについてだけは，チャネル0にD_7〜D_3までの値と$D_0=0$として書き込むことで，D_2, D_1は各チャネルに対応した値がCTCによって自動的に割りふられ設定されます．

● Z80 CTCの動作の制御

一般的なCTCの動作は，図7-9に示すようなフローチャートに従って制御できます．

(1) チャネル制御語

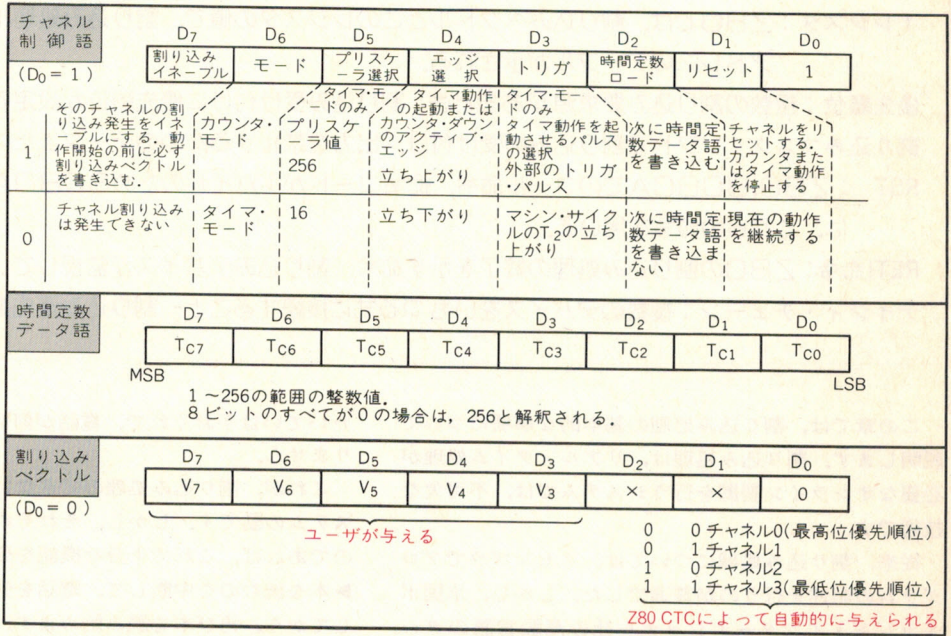
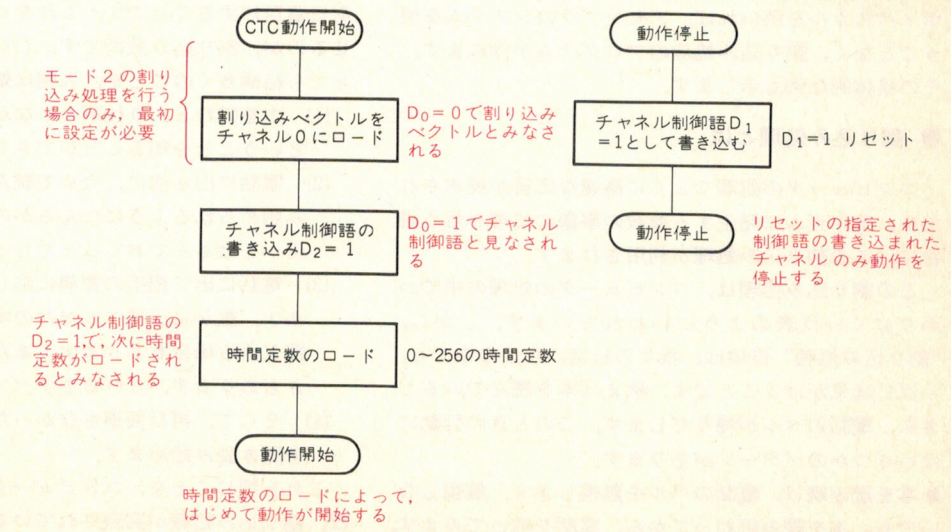
(2) 時間定数のロード

この二つの処理が行われるまでは，CTCの各チャネルは動作を開始しません．

カウンタやタイマは割り込み処理の割り込み源として利用されます．その場合，チャネル制御語のD_7を1とします．このD_7を1に設定したチャネルのカウント・レジスタが0になったときにのみ，割り込みが発生します．

チャネル3は，ZC/TOの出力端子をもっていませんが，割り込み処理の機能はもっています．

CTCの具体的なプログラム例は，次章の割り込み処理の説明のなかで示します．

〈図7-8〉
Z80 CTCのプログラミング

〈図7-9〉
Z80 CTCの動作制御

割り込みのプログラミング

NEXT

割り込み処理とはどういうことかという説明から始まり，ハードウェアとの連携，プログラミングの実例について説明します．

keywords

割り込みベクトル：割り込み時に，割り込みデバイスより出力されるデータ．割り込み処理ルーチンの配置されているアドレスを示すテーブルの位置を示す．

Ｉレジスタ：Z80では，割り込みベクトルとこのレジスタの値で，割り込み処理ルーチンの入口のアドレスのテーブルが示される．

優先順位：複数の割り込み要求源があるとき，割り込み受け付けに優先順位が設定できる．

割り込みマスク：ソフトで割り込みの受け付け．出力を禁止するためのデータまたは処理．

RST　：Z80,8080Aでのコール命令．命令コードが1バイトのため8080Aの割り込み時には，これらの命令を用いる．

RETI命令：Z80の割り込み処理の終了を示す命令．割り込みデバイスが監視している．

デイジィ・チェーン：複数のデバイスをいもづる式に接続すること．割り込み優先順位を決める．

この章では，割り込み処理の基本的な事項について説明します．割り込み処理は，リアル・タイム処理が必要なオンライン制御を行うシステムでは，不可欠な技術です．

従来，割り込み処理については，アセンブラでプログラムの記述を行うのが普通でした．しかし，米国ボーランド・インターナショナル社の高級言語のターボ・パスカルを用いれば，アセンブラのシステムを使うことなく，割り込み処理のプログラムが作れます．その具体的な例も示します．

● 割り込み処理とは

コンピュータの処理でとくに高速な応答が要求されたり，ランダムに発生する複数の事象の処理を行う場合に，この割り込み処理が利用されます．

この割り込み処理は，コンピュータの処理の中でわかりにくい代表のようにいわれています．しかし，“割り込み処理”自体は，我々の日常の生活の中でも，しばしば見かけることです．例えば本を読んでいるときに，電話のベルが鳴りだします．このときの行動にはいくつかのパターンがあります．

▶本を読み続け，電話のベルを無視します．無視しないでも，本を読み終わってから，電話を取ってみます．たいていは手おくれで，電話が何回あったかすらわかりません．

これが，割り込み処理のできないコンピュータ・システムの話です．しかし，それぞれの仕事に専念するのであれば，これで十分な機能を発揮します．

▶本を読むのを中断して，電話を受けその用件を済ましてから，再び本を読み始めます．何のこともない，常日頃目にすることです．これをコンピュータに行わせるのが，割り込み技術です．しかし，これだけのことでも結構多くのことを，人間は処理しています．

(1) 本を読むという仕事をしながら，電話が鳴ったということを知ることができる．

(2) 電話に出る前に，今まで読んでいた本を再び読み始められるようになんらかの印を残す．たぶん何かをはさんでおくことでしょう．

(3) 電話に出て相手の要望に応じた適切な処理をします．誰かに伝言するだけの場合，そのために大騒ぎする場合もあります．また，間違い電話のときもあります．これらをすべて行っています．

(4) そして，再び何事もなかったように同じページを開き読み始めます．

これと同じことを，コンピュータのシステムが行えば，割り込み処理が実現されているといえます．

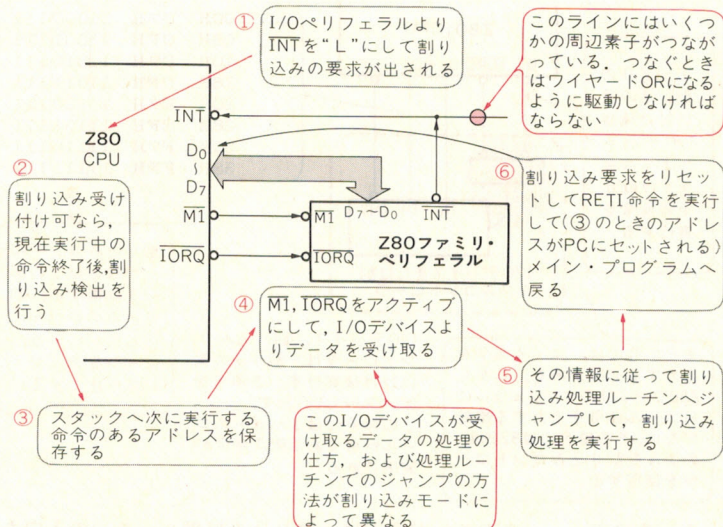
● 具体的に割り込み処理をコンピュータ・システム上で実現するには

Z80のシステムでのこの割り込み処理は，図8-1に示すような流れで行われます．

図8-2に示してあるのは，複数のI/Oデバイスから割り込み要求が発生する場合の，割り込み要求のための信号線の接続方法です．以下，これら割り込み処理の具体的な実現法について説明します．

▶割り込みの検出

この処理の具体的な実現のために，各コンピュータ・システムはそれぞれに工夫がこらされています．まず，なんらかのプログラムの実行中であっても，外部からの処理の要求を検知するための入力信号端子をもっています．この端子が，さきほどの電話のベルの鳴っているのを検出する機能を実現します．

〈図8-1〉 割り込み処理の手順

多くのCPUでは，電源しゃ断のように，いかなる処理にも優先する，緊急な処理のために利用されるノンマスカブルな割り込み処理の端子と，プログラムによって，割り込み処理の受け付けを禁止することのできるマスカブルな割り込み処理の端子の，二つの外部割り込み端子をもつのが一般的です．

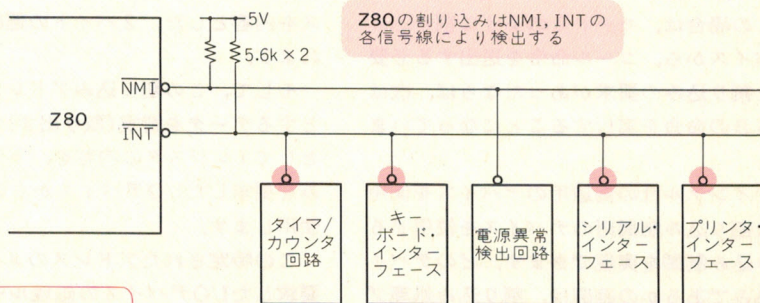
マスク(禁止)可能な割り込み端子は，ディスクの読み込み，通信処理などのように，中断することのできないプログラムの実行中に，外部からの割り込みによって実行が続行できなくなることを防ぐことができます．

▶ プログラムの割り込みによる中断

ハードウェアが割り込みを検出して，割り込みを受け付けた後，現在実行中の処理を中断しなければ，割り込みに対応するプログラムを実行することができません．現在の単一CPUシステムでは，各瞬間ごとでは一つの処理しか実行できません．しかも，自分自身が何をしているかは，知ることができません．せいぜい次に実行する命令を，しかも現在実行中の命令の延長としてしか理解できません．

したがって，中断されていたプログラムを再開する場合も，再開時のステータスの状態を示す，フラグ・レジスタ，再開時に実行すべき命令のあるアドレスがわかれば，再び処理をスタートさせられます．

これらのデータの保存には，スタックを利用します．またシステムによって若干の差はありますが，最小限のデータの保存が自動的に行われます．そのときの状況に応じて，ほかに，保存の必要なデータがあれば，プログラマがソフトウェアで処理し

〈図8-2〉
割り込みを要求するデバイスの接続方法

ます。

このようなことを考慮することで，割り込みによってプログラムが中断しても，割り込み処理の終わった後，中断されていたプログラムの再開が可能となります．

▶ 割り込み処理プログラムの起動法

割り込み要求を受け付けた後，CPUのシステムはその割り込みに応じた処理プログラムを起動させなければなりません．当然のことですが，これら割り込みの処理プログラムも，前もってプログラミングされ，プログラム・メモリ内に書き込まれています．したがって，このルーチンへの制御の移し方を決めればよいことになります．

これを行うには，Z80のシステムでは次のような三つの方法が用意されています．それらは，ハードウェアおよび使用するソフトウェアの状況に応じ，最適なモードを選択することができます．

(1) CALL命令によってジャンプ(モード０)(図8-3,図8-4)．

このモード０は，インテル社の8080A/85などのシステムと同一の割り込みモードです．そのため，インテル社の周辺デバイス，割り込みコントローラを使用する場合は，このモードを選択します．

〈図8-3〉 モード０の割り込み(インテル8080A/85と同じ割り込み)

〈図8-4〉 モード０ではCALL命令を受け取る

(2) 割り込み処理のプログラムのスタート・アドレスが前もって決まっている．そのアドレスからスタートするように，処理のプログラムをコーディングすればよい(モード１)(図8-5)．

このモード１の場合は，モード０のように割り込みを要求したデバイスから，コール命令を送出する必要はありません．割り込みの要求があったならば，次はつねにＲＳＴ３８の命令を実行することになっています．

したがって，インテル社の周辺用のデバイスを使用しても，特別な割り込み制御用のデバイスを使用することなく，割り込み制御を実現できます．どのデバイスからの割り込みであるかの確認は，割り込み処理プログラムの最初に行い，それぞれ要求に応じた処理が実行されます．

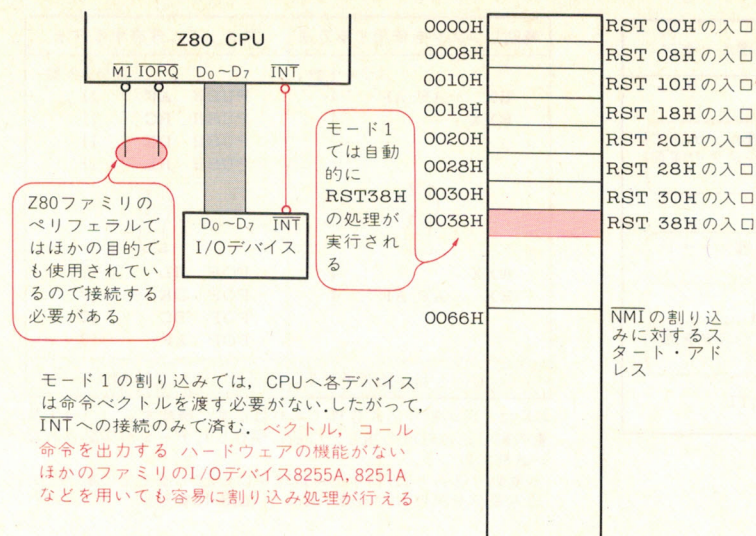
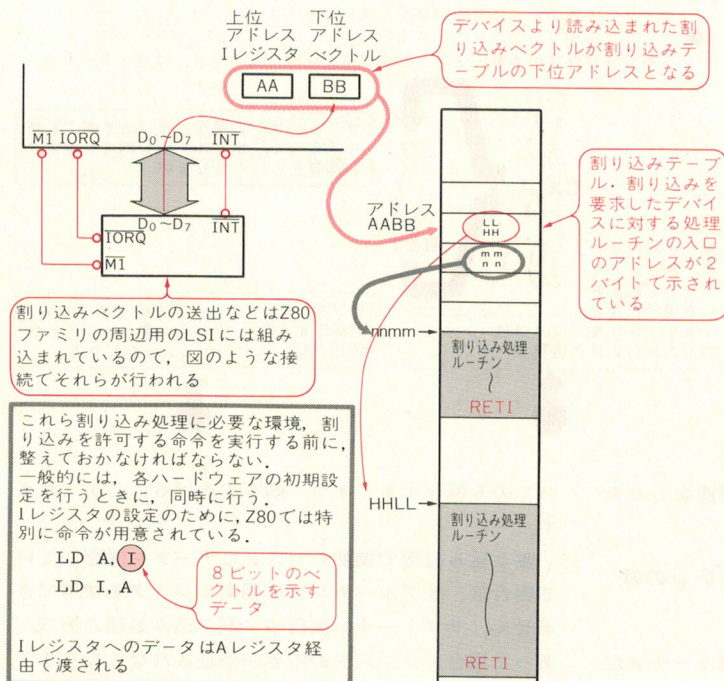
(3) 割り込みベクトルにより，それぞれの要求に応じた処理プログラムに制御が移される(モード２)．

このモード２では，割り込み処理ルーチンの入口を示すためのアドレス・テーブルが用意されます．このテーブルに，各割り込み処理ルーチンの入口のアドレスを内容とした，２バイトの連続したデータが配置されます．

そして，この割り込みアドレス・テーブル中の必要とするデータを特定(取り出す)するには，上位バイトとしてＩレジスタの内容を，下位バイトとして割り込みを要求したI/Oデバイスからの割り込みベクトルを使用します．

この特定されたアドレスのメモリには，割り込みを要求したI/Oデバイスの処理ルーチンのスタート・アドレスが入っています．

モード２の場合は，割り込みを要求するデバイスにZ80のファミリを使用すると，ハードウェアに特別な素子を使用しなくても，割り込み要求に応じた処理プ

〈図8-5〉モード1の割り込み

〈図8-6〉モード2の割り込み

ログラムに直接制御を移すことができます．これにより，プログラムによって割り込みを要求しているデバイスをチェックする必要がなくなり，割り込み処理ルーチンの処理速度を上げることができます(図8-6)．

割り込み処理の具体的な例

割り込み処理によってオペレーションを行う場合は，具体的に次のような手順になります．

(1) 使用する周辺素子，割り込み処理の内容によって，使用する割り込み処理のモードの検討を行う．0～3FHまでの間に割り込み処理のスタート・アドレスが設定できるなら，モード0からモード2まで いずれも使用できる．

(2) 使用するモードに応じて，割り込みコントローラおよび周辺素子の選択を行う．パーソナル・コンピュータなど，すでに完成しているシステムを使用する場合は，そのシステムの中で使用できるモードを検討する．

(3) 使用する割り込みモードが決まったら，次の事項について検討を加える．

割り込みルーチンへのジャンプ命令を所定のメモリへセットする．

▶モード0では，0～38HまでのそれぞれのRST命令のジャンプ先．

▶モード1では，38H一つが割り込み処理ルーチンの入り口となる．

▶モード2では，CPUのIレジスタと周辺装置が割り込み要求時に出力する割り込みベクタから合成されるアドレスが，割り込み処理ルーチンの入口を示すアドレスの入っているテーブルを示す(図8-6参照)．

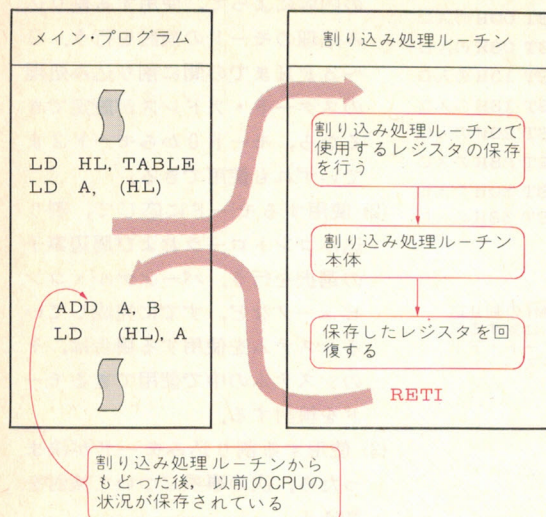
(4) 具体的な割り込み処理ルーチンのコーディングを行う．このとき割り込み処理ルーチンの中で使用するレジスタは，スタックへ保存する必要がある．

● 割り込み処理のプログラムはまず，現在実行中の状態を保存しなければならない

割り込み処理は，現在実行中のプログラムを中断して，全然関係ないプログラムを動かし，その後，中断していたプログラムを再開します．再開時には，中断時のCPUの状態を完全に回復する必要があります．

そのための命令が，Z80には用意されています．しかも，いくつかの方法が選択できます．

もどり番地(PCの値)は，Z80 CPUが割り込みを受け付けたときに，自動的に退避されます．その後の処

〈図8-7〉割り込み処理でのレジスタ，データの保存(1)

〈図8-8〉割り込み処理でのレジスタ，データの保存(2)

補助レジスタを使用する方法	ステート数	スタックに保存する方法	ステート数
EX AF, AF′	4	PUSH AF	11
EXX	4	PUSH BC	11
		PUSH DE	11
		PUSH HL	11
EXX	4	POP HL	10
EX AF, AF′	4	POP DE	10
		POP BC	10
		POP AF	10
合計16ステート		合計84ステート	

EXは1回の割り込み処理であれば正常に機能する．しかし，多重の割り込み処理の場合，以前に保存したデータが保証されない状態がおこる．
多重割り込みとは，割り込み処理中にも，より優先度の高い割り込み要求を受け付け処理すること(図8-10参照)

理に，次に示す二つの処理方法があります．

(1) そのほかのレジスタは，割り込み処理ルーチンの先頭で退避する．

PUSH命令によって，割り込み処理で使用するレジスタをスタックに退避する．割り込み処理中に使用せず，変化しないレジスタは退避する必要はない．

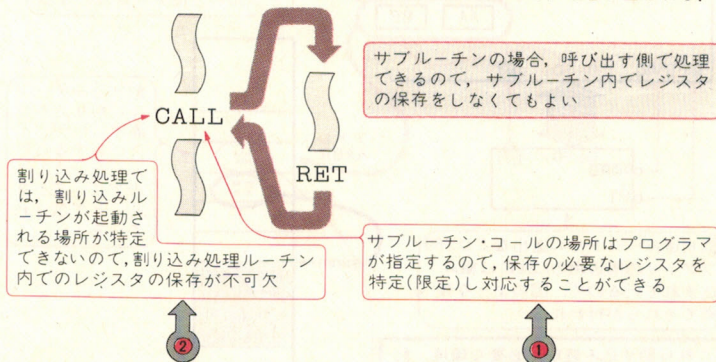
(2) 退避のために，CPUの内部の補助レジスタとの交換命令も用意されている．CPU内部の処理だけなので，外部のメモリとのデータの交換のための時間の必要がなく，高速な処理ができる．

図8-7と図8-8にそれぞれの動作を示します．プログラムの仕様によって，最適なものを選択するようにします．

〈図8-9〉サブルーチンでのレジスタ保存は自由度が大きい

● レジスタの内容の保存は，サブルーチンの処理でも必要となる

レジスタの内容の保存は，割り込み処理ルーチンだけの問題ではなく，通常のサブルーチンの呼び出しでも必要となります．

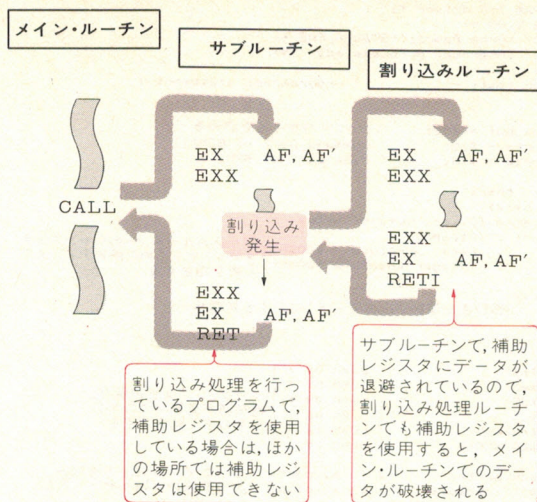
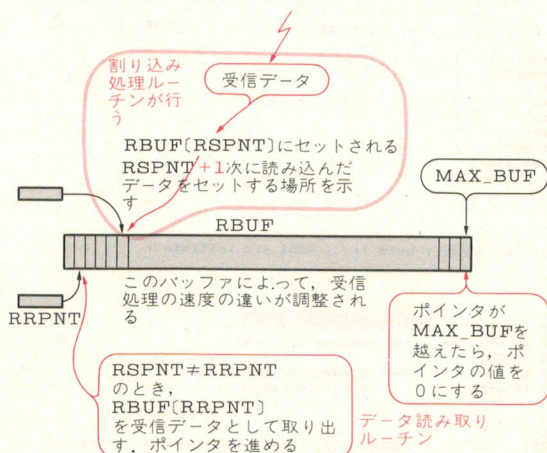
この場合，図8-9に示すように，サブルーチンの内部で保存する場合と，サブルーチンを呼び出すほうで保存する場合があります．プログラムのコーディングのしやすさからいえば，サブルーチン側で保存するほうが，レジスタの保存を考慮せずにプログラミングできます．

プログラム全体の処理速度に厳しい要求がある場合などは，サブルーチンを呼び出す側で必要なものについてのみ保存することで，効率をあげることができます．

割り込み処理で補助レジスタにデータを保存している場合は，サブルーチンでは補助レジスタを使用できません．サブルーチン実行中に割り込み処理の要求があった場合，レジスタの内容が保証されなくなります．そのサブルーチンの実行中に割り込みを禁止する方法もありますが，一般的ではありません(図8-10)．

モード1の割り込みと8251Aを用いたターミナル・モードのプログラム

具体的な割り込み処理のプログラムの例として，シングル・ボード・コンピュータなどのI/O装置となる，コンソール・ターミナルのためのプログラムを作成します．装置としては8251Aを使用したRS-232Cのインターフェースをもっていて，RxRDYによって割り込

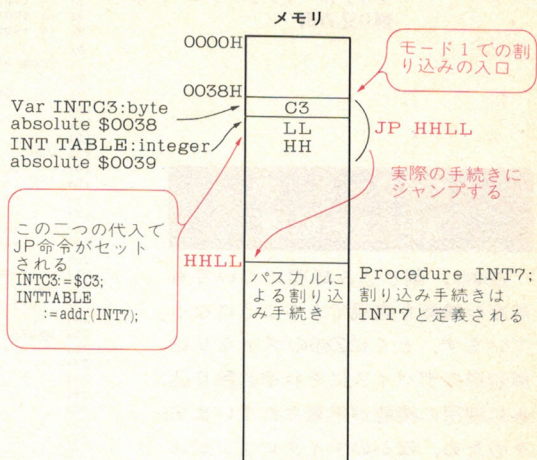
〈図8-10〉レジスタの値の保存の問題点

メイン・ルーチン
サブルーチン
割り込みルーチン
CALL
EX AF, AF'
EXX
割り込み発生
EXX
EX AF, AF'
RET
EX AF, AF'
EXX
EXX
EX AF, AF'
RETI
割り込み処理を行っているプログラムで，補助レジスタを使用している場合は，ほかの場所では補助レジスタは使用できない
サブルーチンで，補助レジスタにデータが退避されているので，割り込み処理ルーチンでも補助レジスタを使用すると，メイン・ルーチンでのデータが破壊される

〈図8-12〉受信データをバッファに入れるときの処理の説明

みがかけられるパーソナル・コンピュータで，Turbo Pascal(Z80用)が動くものなら，8251Aの入出力アドレスを合わせるだけで移植することができます．

従来は，割り込み処理のあるプログラムはアセンブラで記述するのが一般的でしたし，実行スピードも最も高速なむだのないプログラムとなります．しかし，開発と後のメインテナンスのコストを考慮した場合，実行スピードの効率追求のみが真の効率追求とはならなくなっています．

今回使用したTurbo Pascalは，2万円以下で入手可能でありながら，従来からよく使われているPascal MT＋に匹敵する機能をもち，なおインタープリタと同様な使用勝手が得られる高速なコンパイラです．

このTurbo Pascalについては，詳しい説明書も市販されていますので，今回具体的なプログラミングの例を示すにとどめておきます．

〈図8-11〉ジャンプ命令のセット方法

割り込みに関する初期設定は，パスカルの命令で記述できない部分があります．これについては，Inline命令が用意されているので，パスカルの記述の中に直接機械語のコードを16進表示でセットすることにします．

● 割り込みの記述

パスカルで割り込み処理を記述するためには，次のような処理が必要です．

まず，パスカルで記述された割り込み処理ルーチンのアドレスを取り出します．割り込みジャンプ・テーブルに，割り込み処理ルーチンの入口のアドレスをセットします(モード2)．または，割り込み時に使用されるRST命令で呼びだされるエリアに，割り込み処理ルーチンの入口のアドレスをセットします(モード0，1)．モード0，1の場合の具体例を図8-11に示します．

absolute命令を用いて，変数の絶対アドレスを所定のジャンプ命令をセットする場所に指定します．この変数にADR関数によって得られた割り込み手続きのアドレスをセットすることで，上記の目的は達成できます．

アドレス・テーブルは2バイトですから，整数変数だけですみます．ジャンプ命令の埋め込みは，オペコードのC3Hには，BYTE変数を用い，アドレスはintegerの整数変数を用います．

割り込み処理の手続きの中では，標準のI/O手続き，関数は利用できませんので，Inline命令で機械語のコードを直接埋め込んであります．具体的なプログラム・リストをリスト8-1に示しておきます．

なおバッファリングの動作のようすを図8-12に示します．

Z80ファミリの割り込み処理

Z80は，割り込み処理についても高度な機能を実現できるようになっています．とくにZ80のファミリは，周辺用のデバイスにそれぞれ割り込み処理用の機能が用意されています．そのため，ほかのマイクロコンピュータ・ファミリのように，特別な割り込み処理用のデバイスを用いなくても，割り込み処理が行えます．

また，割り込みのモード1においては，各デバイスからの割り込み要求をCPUに出すことができれば，特別なハードウェアを付加することなく割り込み処理が行えます．

したがって，ほかのファミリの素子を用いたシステムでの割り込み処理であっても実現しやすくなっています．

ここでは，これらの具体的な例として，Z80 CTCを用いた割り込み処理の例を説明します．

Z80 CTCを使用してタイマを作成する

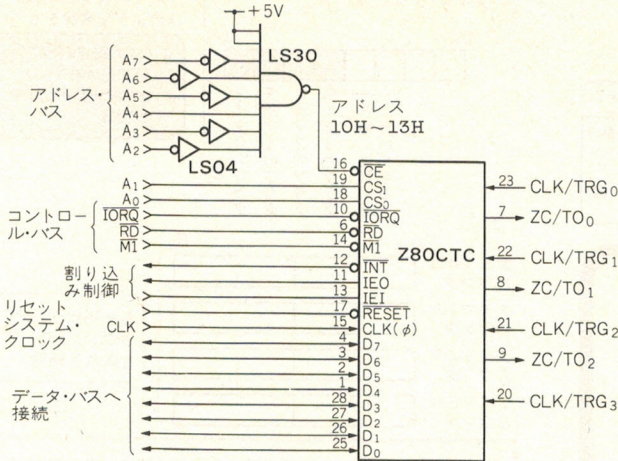
所定の時間ごとにZ80 CTCのタイマより割り込みをかけ，各種のシステムの時間計数の基準とすることができます．このタイマは，図8-13のような構成となります．

Z80 CTCはモード2の割り込みを処理できる機能をもっていますが，比較のため最初にZ80の割り込みモード1を使用します．後でモード2の説明を行います．

Z80のモード1の割り込みは，Z80のファミリ以外の入出力デバイスを用いて割り込み処理を行うときに，大きな効果を発揮します．同じタイマICの8253を利用する場合の考慮点も合わせて説明します．

〈リスト8-1〉
シリアル・インターフェースの割り込み例

```
  1:
  2: (* Test program I/O driver *)
  3: (* 1983/03/26 V.00 *)
  4: (* 1984/05/20  turbo Pascal PSTB010.PAS *)
  5: (* 1985/02/02  interrupt prrocess add    *)
  6:
  7: Program Driver_test;                 { programname statement }
  8:
  9:      Const
 10:            max_buf = 255;      バッファのサイズを決める
 11:            porta = $15;       { command port of 8251 }
 12:            portd = $14;       { data port of 8251 }
 13: Var
 14:     outdat  :  char;
 15:     indata : byte;
 16:     rbuf   : array[0..max_buf] of byte;
 17:     rspnt,rrpnt : integer;
 18:     intc3        :  byte  absolute   $0038;  割り込みルーチンへのジャンプ命令
 19:     inttable     :  integer absolute $0039;  をセットするために絶対アドレスで
                                                  変数を指定する
 20:
 21: procedure int7;
 22: var             RST7または割り込みモード1に対する処理ルーチン
 23:  intflg : byte;
 24:
 25: begin
 26:   inline (  $f5/                { push af }
 27:             $e5/                { push hl }
 28:             $d5/                { push de }   フラグ，レジスタの保存．
 29:             $c5/                { push bc }   機械語が直接パスカルの
 30:             $fd/$e5/            { push ix }   命令の中で記述できる
 31:             $DD/$E5);           { push iY }
 32:
 33:   inline (  $db/porta/          { in A, porta    }  コマンド・ポートを読み，
 34:             $32/intflg);        { ld ( inflag ),A }変数intflgへセットする
 35:   if (intflg and 02) > 0 then
 36:                begin
 37:                  inline ( $db/portd/          { in a,portd }データ・ポートを読み，
 38:                           $32/indata);        { ld (indata),a }Indataセットする
 39:                  rbuf[ rrpnt ] := indata;読み込みデータをバッファに書く
 40:                  rrpnt := rrpnt + 1;    受信バッファの受信データのポインタを進める
 41:                  if rrpnt>max_buf then rrpnt :=0;
 42:                end;  受信バッファの最後を越えた場合，バッファの先頭へポインタを
 43:
 44:   inline (  $DD/$E1/
 45:             $fd/$e1/
 46:             $c1/
 47:             $d1/      フラグ，レジスタをもとに戻してメイン・ルーチンへ戻る
 48:             $e1/
 49:             $f1/
 50:             $FB/
 51:             $ed/$4d);
 52: end;
 53:
 54:
 55:
 56:
 57: Procedure Sioinit( port1 : byte );   { 8251 sio initiate }
 58:
 59:
 60:
 61: Begin
 62:   port[port1] := 00;          8251Aの初期化ルーチン
 63:   port[port1] := 00;   (* dummy command write *)
 64:   port[port1] := 00;
 65:   port[port1] := $40;   (* reset command write *)
 66:   port[port1] := $4E;            (* initiol mode *)
 67:
 68:   rspnt := 0;
 69:   rrpnt := 0;
 70:   intc3 := $C3;             0038H以後に，割り込み処理ルーチンへのジャンプ
 71:   inttable := addr( int7);  命令をセットする
 72:
 73:   inline(     $ED/$56/      { IM 1 }割り込みモード1にセットする
 74:               $FB);         { EI   }割り込み可にする
 75:
 76:
 77: end;
 78:
 79:
 80:
 81: begin
 82:
 83:
 84:   Sioinit(porta);   { usart init. }
 85:
 86:   port[(porta)] := $37;   { 8251 command write }送受信可とする
 87:
 88:   Repeat 未処理の受信データがあるかどうかのチェックをする
 89:
 90:      if rrpnt<>rspnt then
 91:              begin                          受信データがある場合，受信データを
 92:                write( chr(rbuf[ rspnt]) );画面に書き出す．画面へ出力するデー
 93:                rspnt := rspnt +1;          タを示す
 94:                if rspnt > max_buf then rspnt := 0;ポインタを進める．バッファの
 95:             end;                                   最後の場合，ポインタを先頭へ
 96:                                                    セットする
 97:      IF keypressed   then キーボードからの入力をチェックする
 98:        begin
 99:          read(kbd,outdat);
100:          if outdat=chr( 13 ) then write( chr(10));
101:
102:          Repeat
103:          until  (port[porta] mod 2 )=1 ;
104:          port[(portd)] := ord(outdat);
105:        end;
106:    UNTIL ord(outdat)=$1F;
107:    { chr(31) is ^- jump to CP/M monitor }
108:
109: END.              (*   main program end *)
110:
```

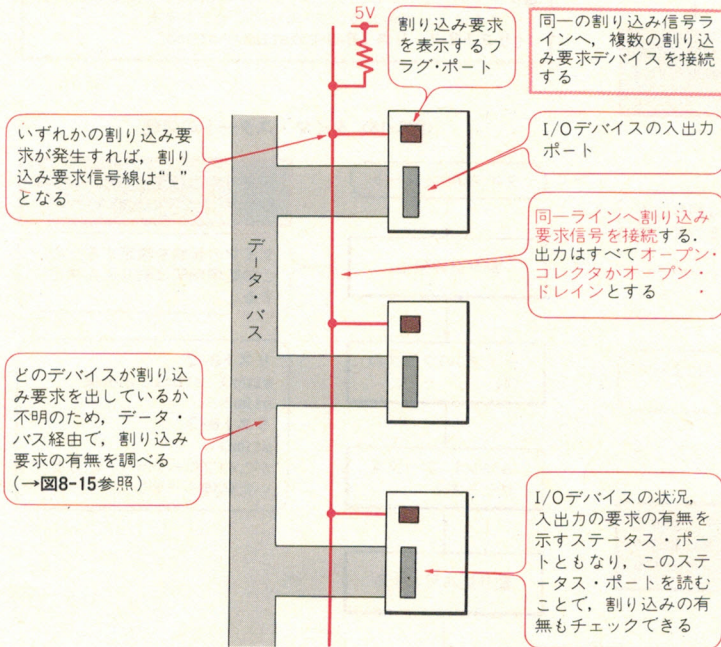
〈図8-13〉 Z80 CTCの接続例

$\overline{CE}$:アドレス・デコードの決め方

① アドレス・バスのうち下位8ビットA_0～A_7にI/Oアドレスが出てくる
② 内部に四つのレジスタがあるので，これを選択するのに2ビット必要．このレジスタは四つのタイマ/カウンタに対応し，それぞれの制御に利用する
③ そのため，チップ・セレクトのCS_0をアドレスのA_0に，CS_1をA_1に接続すると連続に四つのアドレスがとれる
④ そして上位6ビットA_2～A_7で，Z80 CTCのベースとなるI/Oアドレスを**LS30**と**LS04**を組み合わせて000100××Bとデコードする．もちろん，**LS139/138**などでもデコードできる
⑤ その結果は下表のようになり，各チャネルの制御ができる

	CE	CS_1	CS_0	内部レジスタ
10H	000100	0	0	チャネル0
11H	000100	0	1	チャネル1
12H	000100	1	0	チャネル2
13H	000100	1	1	チャネル3

〈図8-14〉 フラグ，ステータス・チェックによる割り込み要求の処理

● モード1で複数の割り込み要求のデバイスがある場合，割り込み処理の優先順位は ソフトウェアで決まる

モード1の場合，どの割り込みが発生したとしてもＲＳＴ３８のコール命令が実行されて，絶対アドレス００３８Ｈ番地からのルーチンの処理に制御が移ります．

このルーチンの中で，まずどのデバイスからの割り込みであるかのチェックを行います．割り込み処理を要求するデバイスの割り込み要求フラグのチェックを行い，割り込みを要求していたならば，そのデバイスに対する割り込み処理を行います．

割り込み要求のチェックを行い，割り込み要求があればその割り込み処理を行うので，この割り込み要求のチェックの順番が割り込み処理の優先順位そのものになります．これらの関係を図8-14で示しておきます．この方法では，どのデバイスからの割り込みであるかを，ソフトウェアで処理する時間が必要となります．

割り込みが要求されてから，割り込み処理が開始されるまでの時間が問題になる場合もあります．しかし，この時間に関する問題がなければ，ほとんどの割り込み処理が，このモードで実現できます．

具体的には図8-15に示すように，ワイヤードＯＲで割り込み要求線に接続されている各デバイスのステータスを，順次調べていきます．その中で，割り込み要求を行ったデバイスから順番に要求に応じた処理を行います．多重割り込み処理を制御するコントローラを使用していない場合は，図8-15に示すように，個々のサービスを終了してから，次の割り込み処理を行うようにします．

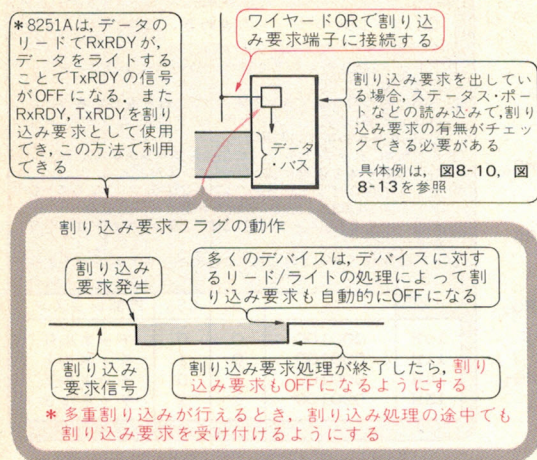
多重割り込みのためには，Z80ファミリのデバイスのもつデイジィ・チェーンの割り込み優先順位の制御を用いるか，インテル系の8259Aのような割り込み制御素子を用います．

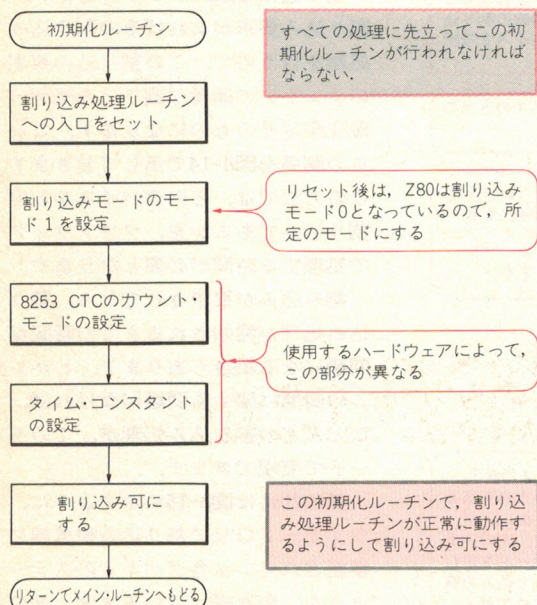
ソフトウェア・タイマを作成して プログラム実行の時間管理ができる

割り込みの具体的な説明として，基本となるタイ

〈図8-15〉割り込み要求デバイスの確認(フラグのチェック)

〈図8-16〉タイマ処理のワーク・エリア

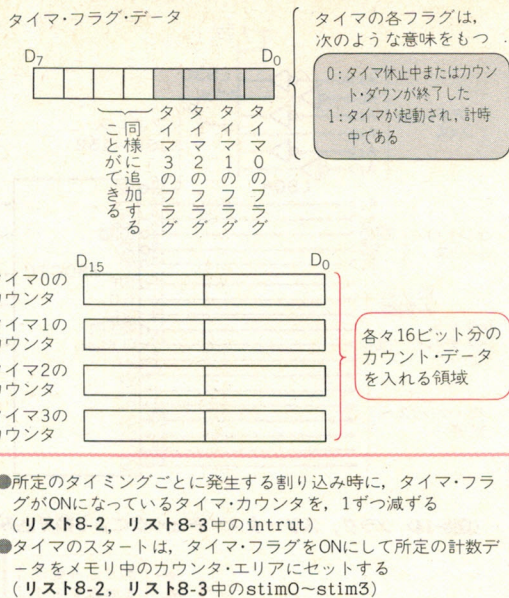
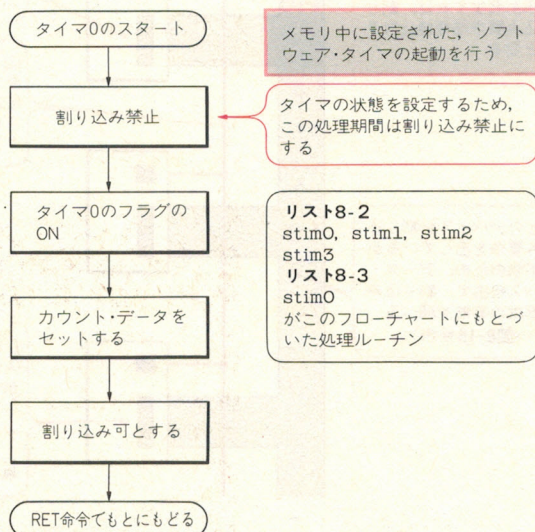
●所定のタイミングごとに発生する割り込み時に,タイマ・フラグがONになっているタイマ・カウンタを,1ずつ減ずる(**リスト8-2**,**リスト8-3**中のintrut)
●タイマのスタートは,タイマ・フラグをONにして所定の計数データをメモリ中のカウンタ・エリアにセットする(**リスト8-2**,**リスト8-3**中のstim0~stim3)

〈図8-17〉8253処理ルーチン・フローチャート

〈図8-18〉タイマ・スタートの処理

ム・ベースをZ80 CTCなどのタイマ用周辺LSIで作成して,それぞれ処理用のソフトウェアのタイマを作成します.

Z80 CTCのタイマからは10msごとに割り込みを発生させて,メモリ上に作成したタイマ用のデータ域の値をカウント・ダウンしていきます.図8-16に示すような方法を用いることで,任意の時間が設定できる万能タイマをソフトウェアで実現できます.

プログラムは次に示すような部分より構成されます.

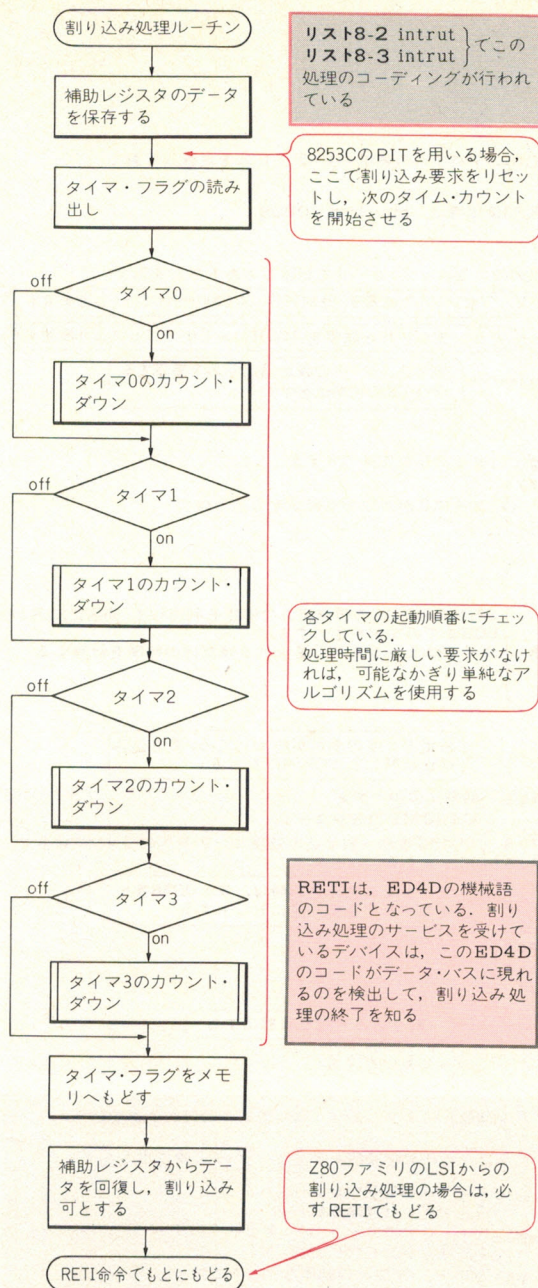
(1) 初期化のルーチン

Z80 CTCの初期化,各ソフトウェア・タイマのデータ域の初期化,割り込み処理ルーチンへの呼び出し命令のセットなどを行う.

(2) タイマのスタートを行う

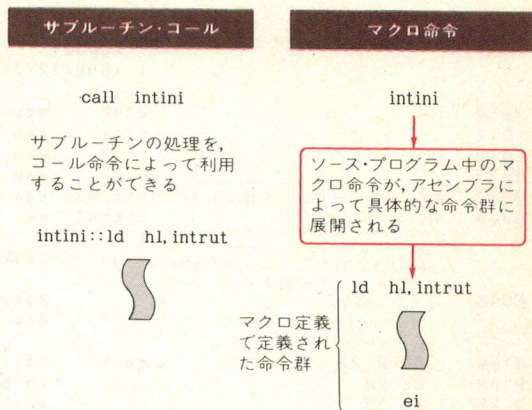
各ソフトウェア・タイマを起動させる処理を行う.このタイマの起動は,アプリケーション・プログラムの任意の場所で簡単な指定で利用できるような工夫が必要.

(3) 割り込み処理ルーチン

ハードウェアの割り込みが発生したとき,それぞれ起動中のソフトウェア・タイマの値をカウント・ダウンしていく.カウント・ダウンの結果,タイマの値が0となったかどうかの確認を行い,0であればカウント・ダウンを停止する処置をとる.

〈図8-19〉タイマのための割り込み処理ルーチン

モード1でのプログラム例を**リスト8-2**に示します．8253を使用する場合の変更点も示しておきます(**図8-17〜図8-19参照**)．

● アプリケーション・プログラムで，任意にプログラミングされた機能を利用できるようにする

タイマの起動ルーチンのようなプログラムは，その機能を用いてより具体的なプログラムを作るための手段となります．そのためには，これらのルーチンが，より具体的なアプリケーション・プログラムのコーディング時に，命令と同等に利用できるような工夫が必要です．

それを具体的に実現する方法が，サブルーチン化とマクロ命令の利用です．サブルーチンの使用は，Z80のコール命令として用意されている機能ですが，マクロ命令とはアセンブラのソース・プログラムを，機械語に変換するアセンブラのシステムに用意された機能です．

〈図8-20〉サブルーチンとマクロ命令

それは，いくつかの基本的な命令を，プログラマ自身が定義したニモニックに置き換えてコーディングする機能です(**図8-20**)．

Z80のマクロ・アセンブラでは，マイクロソフト社のM80というアセンブラが標準となっています．マクロ命令については後で詳しく説明するとして，ここではサブルーチンを使用した例について考えてみます．

● サブルーチンとの間でデータを受け渡す方法

この例の場合，タイマをスタートさせるサブルーチンに汎用性をもたせるために，カウント値をサブルーチンの呼び出し時に指定できるようにします．このとき，このデータの受け渡しが必要になります．

このデータの受け渡し方法は，サブルーチン側と呼び出す側の双方で，その方法を一致させておかなければなりません．この受け渡しの方法は多くの方法が考えられますので，勘違いをしたり，仕様を決めないでプログラムのコーディングをしたために，デバッグ時に苦労することの多い部分です(**図8-21**)．

具体的なデータの受け渡しは，一般的に次のような3種類の方法が利用されています(**図8-22**)．

(1) レジスタ経由でデータの受け渡しを行う．

データの受け渡しに使用するレジスタの種類をあらかじめ決めておき，所定のデータをそのレジスタ

〈リスト8-2〉 Z80のモード1の例

```
                          ; Z8Ø8Ø1
                          ; 1986/12/Ø7
                          ;
ØØ1Ø                      ctcØ     equ 1Øh  ┐
ØØ11                      ctc1     equ 11h  │CTCのデバイスの各ポート・アドレスを定義しておく
ØØ12                      ctc2     equ 12h  │
ØØ13                      ctc3     equ 13h  ┘
ØØB5                      ctc_mod  equ Øb5h ;8253使用時は, ctc_mod=OBOH
ØØC8                      timcst   equ 2ØØ
                          ;
                                   .Z8Ø        Z80のニモニック・コードで記述してあることを示す
                          ;
ØØØØ'                              aseg        アセンブルの結果を,絶対アドレスで割り付けていくことを示す
                                   org 1ØØh    アセンブルされた結果を,100Hのメモリ・アドレスより展開する

Ø1ØØ    CD Ø122           main::   call intini   割り込みモードの設定,割り込みを要求するデバイスの初期化を行う
Ø1Ø3    ØE 2Ø                      ld c,2Øh
Ø1Ø5    Ø6 3Ø                      ld b,3ØH
Ø1Ø7    21 ØØ1E           rp:      ld hl,3Ø
Ø1ØA    CD Ø18C                    call stimØ      ;タイマ0をスタートする
Ø1ØD    3A Ø1C4           lp:      ld a,(timf) ┐
Ø11Ø    B7                         or a        │;タイマ0がカウント終了するのを待つ
Ø111    2Ø FA                      jr nz,lp    ┘
                          ;
Ø113    ØC                         inc c       ┐
Ø114    C5                         push bc     │
Ø115    59                         ld e,c      │CP/M80のBDOSの標準出力の機能を利用して,画面に21Hから
Ø116    ØE Ø2                      ld c,2      │50Hまでの文字を出力する.
Ø118    CD ØØØ5                    call ØØØ5h  │タイマはHLレジスタで渡される値だけの時間を計時する
Ø11B    C1                         pop bc      │
Ø11C    1Ø E9                      djnz rp     ┘
                          ;
Ø11E    F3                         di
Ø11F    C3 ØØØØ                    jp ØØØØh      所定の文字の表示が終わったら,割り込み禁止状態にしてCP/Mにもどる
                          ;
Ø122    21 Ø139           intini::ld hl,intrut   初期化のルーチン
Ø125    3E C3                      ld a,ØC3H       ;C3はJMPの命令コード
Ø127    32 ØØ38                    ld (ØØ38H),a  ┐0038H番地へ割り込み処理ルーチンへのジャンプ命令を
Ø12A    22 ØØ39                    ld (ØØ39H),hl ┘セットする
                          ;
Ø12D    ED 56                      im 1          モード1の割り込み時は,常に0038番地をコールするモードにセットする
                          ;
Ø12F    3E B5                      ld a,ctc_mod    CTCの初期化を行う
Ø131    D3 13                      out (ctc3),a
Ø133    3E C8                      ld a,timcst
Ø135    D3 13                      out (ctc3),a
Ø137    FB                         ei            割り込み可能な状態にする
                          ;
Ø138    C9                         ret             ;コール先にもどる
                          ;;
Ø139    Ø8    割り込み処理ルーチン intrut::ex af,af' ┐
Ø13A    D9                         exx           ┘補助レジスタにより,各レジスタの値の保存を行う
Ø13B    3A Ø1C4                    ld a,(timf)   ┐
Ø13E    57                         ld d,a        ┘タイマの起動状態を示すフラグ・データを取り出す
Ø13F    CB 42                      bit Ø,d         ;タイマ0が起動中か調べる
Ø141    28 ØD                      jr z,nx1        ;停止中であれば次を調べる
Ø143    2A Ø1C5                    ld hl,(timØ)    ;タイマ0のカウント・データを取り出す
Ø146    2B                         dec hl          ;1だけカウント・ダウンする
Ø147    22 Ø1C5                    ld (timØ),hl    ;カウント・ダウンの結果を元にもどす
Ø14A    7C                         ld a,h        ┐
Ø14B    B5                         or l          ┘カウント・ダウンの結果がゼロかどうか調べる
Ø14C    2Ø Ø2                      jr nz,nx1       ;ゼロでなければ次のタイマ1に進む
Ø14E    CB 82                      res Ø,d         ;ゼロならフラグをリセットする
Ø15Ø    CB 4A             nx1:     bit 1,d         ;タイマ1を調べる
Ø152    28 ØD                      jr z,nx2
Ø154    2A Ø1C7                    ld hl,(tim1)
Ø157    2B                         dec hl
Ø158    22 Ø1C7                    ld (tim1),hl
Ø15B    7C                         ld a,h        ┐
Ø15C    B5                         or l          ┘
Ø15D    2Ø Ø2                      jr nz,nx2
Ø15F    CB 8A                      res 1,d
Ø161    CB 52             nx2:     bit 2,d
Ø163    28 ØD                      jr z,nx3
Ø165    2A Ø1C9                    ld hl,(tim2)
```

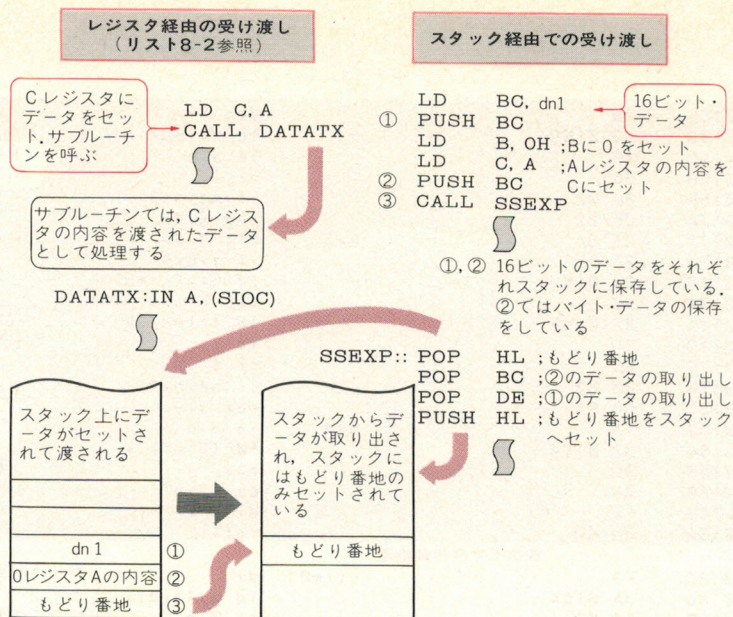
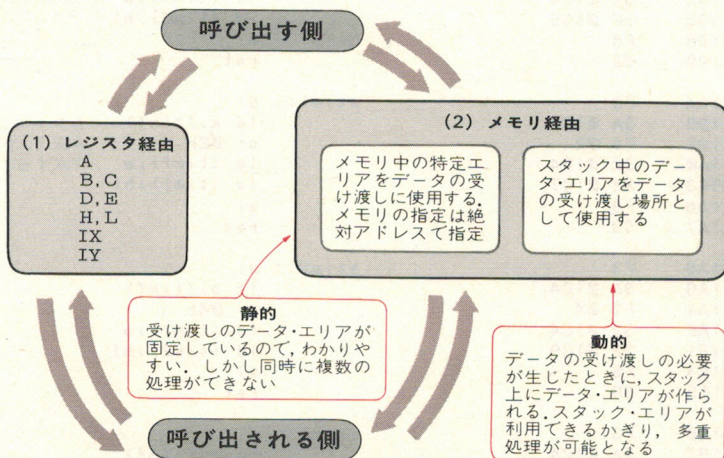
Annotations:

- asegで絶対アドレスを指定しているので'がない
- 呼び出されたサブルーチンが絶対アドレスで定義されているので,'がついていない
- 16ビットのデータは,メモリ上は下位,上位の順になっている.M80のアセンブル・リストでは,上位,下位の順になっている
- HLレジスタのゼロのチェックは,上位,下位共にゼロのときだけ,"H"or"L"がゼロになることにより調べる
- 16ビットのデータをメモリと受け渡しする場合,対象がHL,IY,IXとなるので,HLレジスタを用いる

```
Ø168    2B                        dec hl
Ø169    22 Ø1C9                   ld (tim2),hl
Ø16C    7C                        ld a,h        } ←"H"or"L"の命令がないので,いったん"H"を
Ø16D    B5                        or l          }   Aに移し,A or Lを調べる
Ø16E    2Ø Ø2                     jr nz,nx3
Ø17Ø    CB 92                     res 2,d
Ø172    CB 5A           nx3:      bit 3,d
Ø174    28 ØD                     jr z,nx4
Ø176    2A Ø1CB                   ld hl,(tim3)
Ø179    2B                        dec hl
Ø17A    22 Ø1CB                   ld (tim3),hl  } タイマの数を同様なアルゴリズムで 8個まで ふやす
Ø17D    7C                        ld a,h        } ことができる
Ø17E    B5                        or l
Ø17F    2Ø Ø2                     jr nz,nx4
Ø181    CB 9A                     res 3,d
Ø183    7A              nx4:      ld a,d        ←Dレジスタに各タイマの状態がセットされている
                                                  ので,メモリへもどす.Aレジスタ経由でメモリ
                                                  へもどす
Ø184    32 Ø1C4                   ld (timf),a
Ø187    D9                        exx           } 補助レジスタに保存されていた割り込み時のレジスタの
Ø188    Ø8                        ex af,af'     } 値を回復する
Ø189    FB      割り込み可にする→ ei
Ø18A    ED 4D                     reti          必ず割り込み処理からもどるときはRETIでもどる
                各タイマの起動処理 ;
Ø18C    F3              stimØ::   di            ;割り込み禁止にする
Ø18D    3A Ø1C4                   ld a,(timf)   } タイマ・フラグを取り出し,タイマの起動を示すフラグを1
Ø19Ø    F6 Ø1                     or Ø1h        } にしてメモリへもどす
Ø192    32 Ø1C4                   ld (timf),a   } 経時時間を示すカウント・データをHLレジスタより得て,
Ø195    22 Ø1C5                   ld (timØ),hl  } タイマ0のカウント・データとしてメモリへセットする
Ø198    FB                        ei            } ←割り込み可にして元にもどる
Ø199    C9                        ret           }
                        ;
Ø19A    F3              stim1::   di
Ø19B    3A Ø1C4                   ld a,(timf)   各タイマごとにフラグのビット・カウントする格納エリアの
Ø19E    F6 Ø2                     or Ø2H        み変え, 同様なアルゴリズムでそれぞれの起動ルーチンを
Ø1AØ    32 Ø1C4                   ld (timf),a   作成する
Ø1A3    22 Ø1C7                   ld (tim1),hl
Ø1A6    FB                        ei
Ø1A7    C9                        ret
                        ;
Ø1A8    F3              stim2::   di
Ø1A9    3A Ø1C4                   ld a,(timf)
Ø1AC    F6 Ø4                     or Ø4h
Ø1AE    32 Ø1C4                   ld (timf),a
Ø1B1    22 Ø1C9                   ld (tim2),hl
Ø1B4    FB                        ei
Ø1B5    C9                        ret
                        ;
Ø1B6    F3              stim3::   di
Ø1B7    3A Ø1C4                   ld a,(timf)
Ø1BA    F6 Ø8                     or Ø8H
Ø1BC    32 Ø1C4                   ld (timf),a
Ø1BF    22 Ø1CB                   ld (tim3),hl
Ø1C2    FB                        ei
Ø1C3    C9                        ret
                        ;
Ø1C4    ØØ              timf::  db  Ø           タイマ・フラグ・データ
Ø1C5    ØØØØ            timØ::  dw  Ø           タイマ0のカウント・データの格納エリア
Ø1C7    ØØØØ            tim1::  dw  Ø           タイマ1のカウント・データの格納エリア
Ø1C9    ØØØØ            tim2::  dw  Ø           タイマ2のカウント・データの格納エリア
Ø1CB    ØØØØ            tim3::  dw  Ø           タイマ3のカウント・データの格納エリア

                        ;
                                  end           アセンブラ・ソースの終わりを示す
```

```
D>A:ZSID Z80801.COM
ZSID VERS 1.4
NEXT  PC  END
0200 0100 A9FF
#-D100,200
0100: CD 22 01 0E 20 06 30 21 1E 00 CD 8C 01 3A C4 01
0110: B7 20 FA 0C C5 59 0E 02 CD 05 00 C1 10 E9 F3 C3
0120: 00 00 21 39 01 3E C3 32 38 00 22 39 00 ED 56 3E
0130: B5 D3 13 3E C8 D3 13 FB C9 08 D9 3A C4 01 57 CB
0140: 42 28 0D 2A C5 01 2B 22 C5 01 7C B5 20 02 CB 82
```

下位,上位の順になっている

〈図8-21〉
サブルーチンとのデータの受け渡し

〈図8-22〉
データの受け渡し方法の具体例

にセットし受け渡しを行う．8ビット/16ビットのデータの受け渡しによく利用される．CPU内部のレジスタを利用するために，受け渡しの処理が迅速に行える．

(2) メモリにデータ・エリアを設定し，受け渡しを行う．

この方法は，データ・エリアの設定の仕方で，次の二つの方法があります．

(a) 各処理プログラムから共通に利用できるデータ・エリアを設定し，そのデータ・エリアに受け渡しのデータをセットする．

(b) データの受け渡しが，1対1で対応している場合，スタックを用いたデータの受け渡しが行える．この方法は，データの受け渡しの必要が生じたときに，スタック上にデータの受け渡しのエリアを設定し，処理の制御を相手側に渡す．普通はコール命令が使用される．

そして，そのデータの受け渡しの処理が終わると，データ・エリアがスタックから削除されます．高級言語の処理ルーチンとの間のデータの受け渡しには，この方法が用いられています(図8-23)．

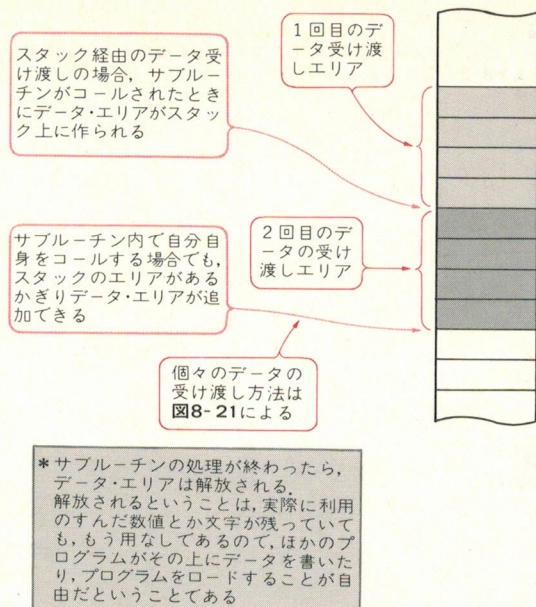
次に，具体的なプログラム例を示して説明します．

モード2の割り込みでは，割り込み受け付け後，直接その処理ルーチンへジャンプする

Z80のモード2での割り込み処理では，CPUが割り込みを受け付けると，割り込みを要求したデバイスから割り込みベクトルを受け取ります．この割り込みベクトルというのは，割り込み処理ルーチンの入り口のアドレスがセットされているテーブルのアドレスの下位バイトを示します．

前にも述べたように，このアドレス・テーブルの上

〈図8-23〉 スタックによるサブルーチンとのデータの受け渡し

〈図8-24〉 Z80割り込みモード2の設定

位アドレスは，CPU内部のIレジスタにセットされている値になります．プログラマは，このレジスタに任意の値をセットすることができます．したがって，割り込みアドレス・テーブルはメモリの任意の場所に設定できます．モード0，1などにくらべ，プログラムの自由度が増しています．

また，割り込み処理ルーチンの最初に，どのデバイスからの割り込みであるかのチェックをする必要がなく，**直接それぞれの処理ルーチンの制御に移れるので，割り込み処理の起動が高速に行えます．**

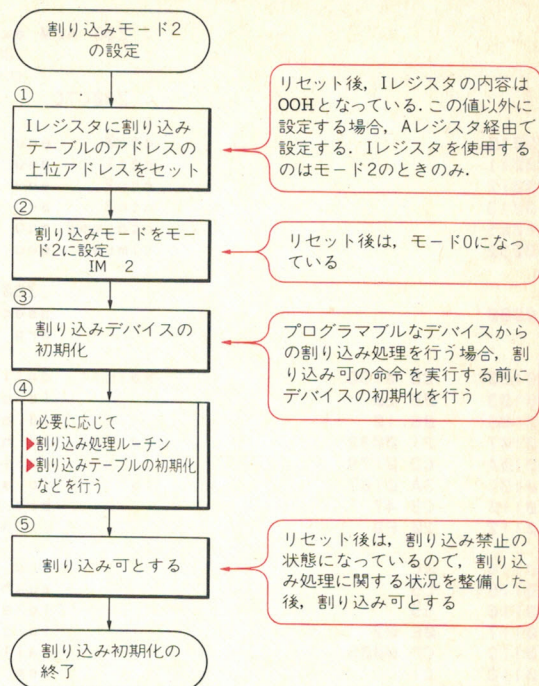
● Z80ファミリの周辺デバイスでは，特別のハードは必要とせずに割り込み処理が実現できる

Z80ファミリの周辺デバイスは，モード2での割り込み処理のための機能がハードウェアで用意されています．これは割り込みの要求後，CPUからの割り込みベクトルの読み込みに対応して，任意のベクトルを送出する機能をもっています．

このデバイスが送出するベクトルは，周辺デバイスの初期設定時に，初期化プログラムによって設定されます．したがって，Z80のモード2で割り込み機能を利用するときには，特別なハードウェア素子を必要とすることなく実現できます．

● Z80 CTCでモード2の具体的なプログラムを考える

最初に割り込み処理関係の初期化を行います．

モード2での割り込み処理を実現するには，図8-24に示すような手順にしたがって処理します．

割り込みを受け付け可にする前に，

① CPU内のI(割り込み)レジスタに，割り込みテーブルのアドレスの上位バイトをセットする．

```
ld a,adl
ld i,a
```

(adlはアドレスの上位バイト)

② 割り込みモードの設定

③ 周辺機器のデバイスの初期設定

④ 割り込み処理ルーチン・プログラムの初期化，ロードが必要な場合はそれらの処理を行う．これにより，いつ割り込み要求があっても暴走することなく処理ができるようにする．

⑤ CPUの割り込み許可フラグを割り込み可の状態にする．

これは**EI命令の実行で行う．**

デバイス側でも，割り込み処理を行うか否かの指定を行えるようになっています．このデバイス側の割り込み許可のフラグは，以上の処理が終わった後，デバイス側の必要に応じてON/OFFし，不必要な割り込み要求がデバイス側から出ないようにします．

● CTCの初期化処理は，各チャネルごとにコマンド・ワードを書き込む

Z80 CTCの初期化は，それぞれのチャネルにコマンド・ワードを書き込むことで行われます．デバイスを制御するためのポートはありません．それぞれのポ

〈リスト8-3〉 Z80の割り込みモード２の例

```
                              ; Z8Ø ASM exsample
                              ;
                              ;
                              ; Z8ØCTC
                              ;
ØØ1Ø                          ctcØ      equ 1Øh
ØØ11                          ctc1      equ 11h
ØØ12                          ctc2      equ 12h
ØØ13                          ctc3      equ 13h
ØØB5                          ctc_mod   equ Øb5h
ØØ32                          timcst    equ 5Ø
                              ;
                                        .Z8Ø
ØØØØ'                                   aseg
                                        org 1ØØh
                              ;
Ø1ØØ      CD Ø13D             main::    call intini
Ø1Ø3      ØE 2Ø                         ld c,2Øh
Ø1Ø5      Ø6 1Ø                         ld b,1ØH
Ø1Ø7      21 ØØ32             rp:       ld hl,5Ø
Ø1ØA      CD Ø179                       call stimØ
Ø1ØD      3A Ø187             lp:       ld a,(timf)
Ø11Ø      CB 47                         bit Ø,a
Ø112      2Ø F9                         jr nz,lp
                              ;
Ø114      ØC                            inc c
Ø115      C5                            push bc
Ø116      59                            ld e,c
Ø117      ØE Ø2                         ld c,2
Ø119      CD ØØØ5                       call ØØØ5h
Ø11C      C1                            pop bc
Ø11D      1Ø E8                         djnz rp
                              ;
Ø11F      Ø6 1Ø                         ld b,1ØH
Ø121      21 ØØ14             rp1:      ld hl,2Ø
Ø124      CD Ø179                       call stimØ
Ø127      3A Ø187             lp1:      ld a,(timf)
Ø12A      CB 47                         bit Ø,a
Ø12C      2Ø F9                         jr nz,lp1
                              ;
Ø12E      ØC                            inc c
Ø12F      C5                            push bc
Ø13Ø      59                            ld e,c
Ø131      ØE Ø2                         ld c,2
Ø133      CD ØØØ5                       call ØØØ5h
Ø136      C1                            pop bc
Ø137      1Ø E8                         djnz rp1
                              ;
Ø139      F3                            di
Ø13A      C3 ØØØØ                       jp ØØØØh
                              ;
Ø13D      3E Ø1               intini::  ld a,inttb  shr 8
Ø13F      ED 47                         ld i,a
                              ;
Ø141      ED 5E                         im 2
                              ;
Ø143      3E 9Ø                         ld a,ctciØ and ØØffh
Ø145      D3 1Ø                         out (ctcØ),a
Ø147      3E 31                         ld a,31h
Ø149      D3 1Ø                         out (ctcØ),a
Ø14B      D3 11                         out (ctc1),a
Ø14D      D3 12                         out (ctc2),a
Ø14F      3E B5                         ld a,ctc_mod
Ø151      D3 13                         out (ctc3),a
Ø153      3E 32                         ld a,timcst
Ø155      D3 13                         out (ctc3),a
Ø157      FB                            ei
                              ;
Ø158      C9                            ret
```

Z80のモード２の割り込みの例．Z80CTCをタイマとするルーチン全体の処理は，**リスト8-2**と同様である

$D_{15}D_{14}D_{13}D_{12}D_{11}D_{10}D_9D_8D_7\ D_6\ D_5\ D_4\ D_3\ D_2\ D_1D_0$

↓

$0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ D_{15}D_{14}D_{13}D_{12}D_{11}D_{10}D_9D_8$

この数だけシフトする

割り込み処理ルーチンのアドレス･テーブルの上位アドレスを得る

CTCのタイマ０用の割り込みテーブルの下位アドレスを得て，割り込みベクトルとしてセットする

〈リスト8-3〉つづき

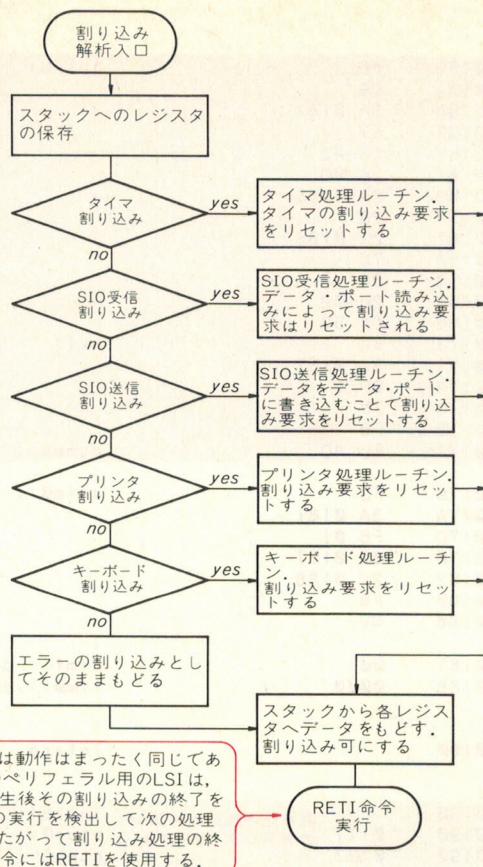
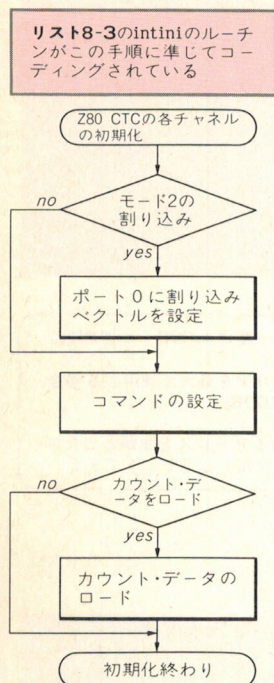
```
                              ;;
Ø159   Ø8                     intrut::ex af,af'
Ø15A   D9                             exx
Ø15B   3A Ø187                        ld a,(timf)
Ø15E   57                             ld d,a
Ø15F   CB 42                          bit Ø,d
Ø161   28 ØD                          jr z,nx1
Ø163   2A Ø188                        ld hl,(timØ)
Ø166   2B                             dec hl
Ø167   22 Ø188                        ld (timØ),hl
Ø16A   7C                             ld a,h
Ø16B   B5                             or l
Ø16C   2Ø Ø2                          jr nz,nx1
Ø16E   CB 82                          res Ø,d
Ø17Ø   7A                     nx1:    ld a,d
Ø171   32 Ø187                        ld (timf),a
Ø174   D9                             exx
Ø175   Ø8                             ex af,af'
Ø176   FB                             ei
Ø177   ED 4D                  dummy:: reti
                              ;
Ø179   F3                     stimØ:: di
Ø17A   3A Ø187                        ld a,(timf)
Ø17D   F6 Ø1                          or Ø1h
Ø17F   32 Ø187                        ld (timf),a
Ø182   22 Ø188                        ld (timØ),hl
Ø185   FB                             ei
Ø186   C9                             ret
                              ;
Ø187   ØØ                     timf:: db  Ø
Ø188   ØØØØ                   timØ:: dw  Ø

                              ;
Ø19Ø                          intare  equ ØfffØh and ( $ + 1Øh )
                                      org intare

Ø19Ø                          inttb::
Ø19Ø   Ø177                   ctciØ:  dw  dummy
Ø192   Ø177                   ctci1:  dw  dummy
Ø194   Ø177                   ctci2:  dw  dummy
Ø196   Ø159                   ctci3:  dw  intrut


                                      end
```

割り込み処理ルーチンのアドレス・テーブル

割り込みテーブルは，一度に*nn*OOH～*nn*FFHにセットできる．
割り込みテーブルのエリアを最大で使用する場合，
OFFOOH AND ($＋100H)
で計算する．
下位ビットがOOHとなるアドレスを先頭としたエリアが，割り込みテーブルとしてセットできる

〈図8-26〉
割り込み解析ルーチン

このような割り込み解析ルーチンを用意することで，特別なハードウェアの必要なく，各種の割り込み処理を実現することができる．
割り込みモード1に設定する．
割り込み優先順位は，解析処理ルーチンの入口に近いほど高くなっている．
このように解析ルーチンの中で，位置によって自由にそれぞれの処理ルーチンの優先順位をコントロールすることができる

〈図8-25〉 Z80 CTCの初期化

ートに，所定の仕様のコマンド・ワードを書き込むことで各チャネルの制御を行います(**リスト8-3,図8-25**参照)．

コマンドとデータなどの区別は，D_0ビットと，コマンドおよびデータの書き込みの順番によって決まります．

D_0が1のときはコマンドと解釈され，0のときは割り込みベクトルの設定と解釈されます．ただし，このベクトルの$D_{1,2}$ビットは，CTCのチャネルに対応したものになります．

データとして書き込む必要があるのは，カウンタ/タイマともに，カウント・ダウンするための定数の設定です．この定数の設定はコマンドのD_2ビットを1にして，次に時間定数を設定すると，指定を行った後に書き込まれたものが定数と解釈されます．

これらの書き込みの順番を間違えると正しい動作が行われず，ハードがおかしいのではないかと戸惑う場合があります．

● Z80ファミリのデバイスからの割り込み処理ルーチンの終了にはRETIが必要

Z80ファミリの周辺用のデバイスからの割り込み処理ルーチンを記述する場合，Z80では，割り込み処理の終了を示す特別なリターン命令を使用します．

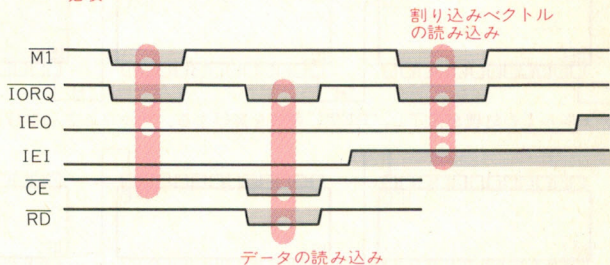
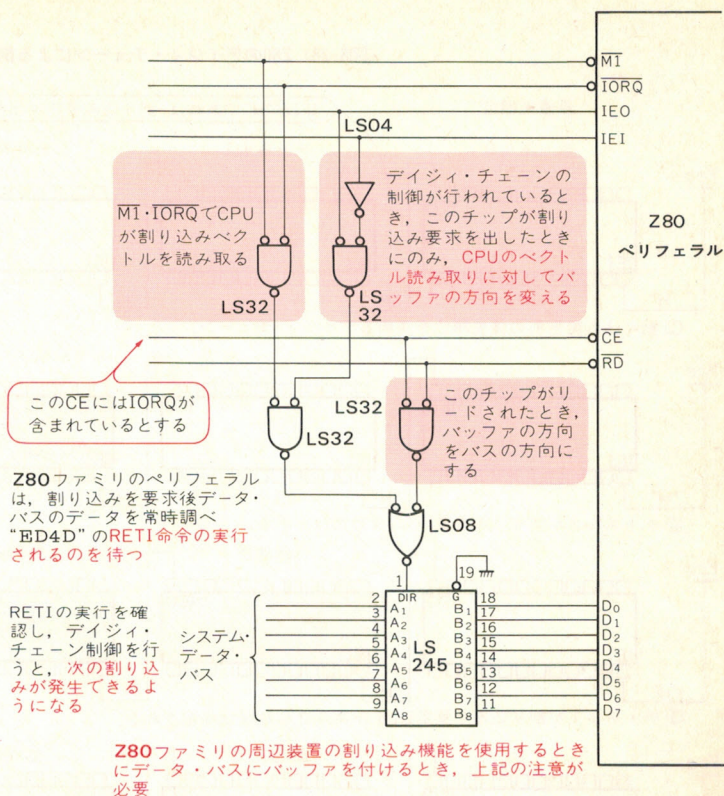
インテル社の8080A/85では，割り込み処理ルーチンからメイン・ルーチンへもどる場合，通常のCALL命令で呼ばれたサブルーチンの終了を示すRET命令でもどります．割り込み処理ルーチンは，サブルーチンとして記述すればすみます．

しかし，Z80の周辺デバイスを用いたシステムでは，通常のサブルーチンと，割り込み処理によって起動された割り込み処理のためのルーチンは区別されます．

これは，割り込みを発生したデバイスが割り込み処理の終了を監視していて，CPUが実行する命令から要求した割り込み処理の終了を知り，次の割り込み発生のための準備を行うためです．

● Z80のデバイスは割り込みの優先順位の制御も行う

複数のI/Oデバイスから割り込みが要求された場合，その緊急度に応じて割り込み受け付けに，優先順位を決める必要が生じる場合があります．図8-2に示すような配線の場合は，図8-26に示すようにソフトウェアで処理することもできます．

〈図8-27〉
割り込み機能使用時のバッファ・コントロール

Z80の周辺デバイス間，またはデバイス内で複数の割り込み要求が同時に発生する場合があります．要求度の高い割り込み処理の要求に対して，CPUは優先して処理をする必要があります．

Z80では，この割り込み要求の優先順位付けを，周辺デバイスの接続順番で処理できるようになっています．優先順位のより高いデバイスを先頭に，優先順位にしたがって順番にデバイスを接続していきます．

この制御に利用される2種類の制御線は，次のような役割を担っています．

▶IEI(Interrupt Enable In)

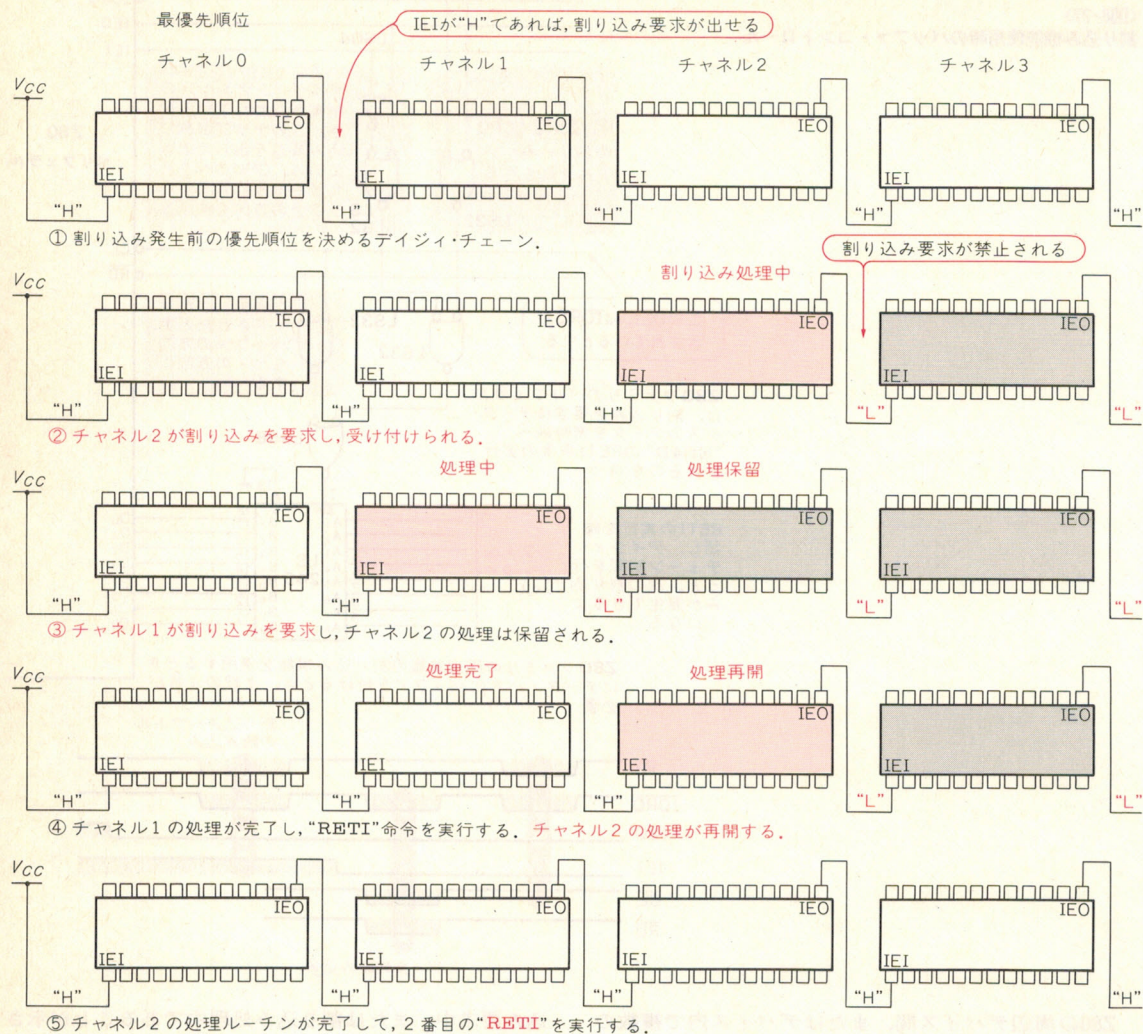
この信号線が"H"のとき，このデバイスより優先順位の高い割り込みが生じていないことを示します．したがってIEIが"H"となっているデバイスは，CPUに対して割り込みを要求することができます．

▶IEO(Interrupt Enable Out)

IEIが"L"でより優先順位の高いデバイスが割り込み要求中，または割り込み処理中であることが示されます．これに応じてより優先順位の低いデバイスに対して割り込みの要求を禁止するためにIEOを"L"にします．また，自分自身が割り込み要求，あるいは処理中もIEOを"L"にして，ほかのデバイスの割り込みを抑制します．

この接続は1本の割り込み制御ラインでつながれ，優先順位の高いデバイスの出力が，次の順位のデバイスの入力となるように接続されています．自分より優先順位の高いデバイス，または自分自身が割り込み要求を出しCPUからの割り込み処理サービスを受けているデバイスは，優先順位の下位のデバイスに対し出力を"L"にして割り込み要求を抑制します．

各デバイスは，割り込み制御ラインの入力端子が"H"でなければ割り込み要求が出せません．優先順位の最も高いデバイスの割り込み制御ラインの入力は，V_{CC}に接続しておきます．

〈図8-28〉 Z80のデイジィ・チェーンによる割り込み処理

これら優先順位の処理は，すべて周辺デバイス側で行います．割り込み処理のサービスを受けているデバイスは，CPUの割り込み処理が終了するのを監視し続けます．

割り込み処理の終了は，RETI命令が実行されたことを検出して知ります．割り込み処理が終了したことが判明した時点で，割り込み要求の抑制を解き，下位の優先順位の割り込み要求を出せるようにします．

RETI命令の検出は，データ・バス上にプログラム・メモリから読み込まれる命令コードを，周辺デバイスが監視することで行っています．このため，周辺デバイスが常にデータ・バスのデータを読めるようにしておく必要があります．バス・バッファなどで，方向の制御を行っているときは，注意しなければなりません．

また，周辺デバイスへ割り込み処理の終了を知らせるために，割り込み処理ルーチンの最後は割り込みからのリターンを示すRETI命令を使用します(図8-27)．

このように，割り込みの優先順位付けは，図8-28に示すように1本のラインで制御され，優先順位の高い順番に接続するだけのシンプルでわかりやすい仕組みになっています．

上級プログラミング

■ NEXT

ソフトウェアを蓄積して，より効率的なプログラミングを行うためには，リンカ，ライブリアン，マクロ命令を自由に使えなければなりません．

keywords

マクロ命令：複数のアセンブラ命令の代わりに記述し，コーディングの効率を上げるための機能．
M80　：マクロ命令の機能をもったZ80用の代表的なマクロ・アセンブラ．
CP/M　：Z80，8080A用の代表的なDOS．Z80などのシステムの開発に利用されている．
DOS　：マイコン・システムではフロッピ・ディスクのためのオペレーティング・システムのことを示す．
コンソール：DOSなどのシステムとの入出力を行うための装置．キーボード，ディスプレイ．
ラベル：プログラム上で，アドレス，データの数値の代わり使用される名札．
ローカル：システム全体でなく，サブルーチン，モジュールなどの対象となる局所を指す
DDT　：CP/M付属のデバッグのためのツール．8080Aが対象．
ZSID　：Z80用でデバッグにラベルを使用することもできるデバッグ・ツール．

● マクロ命令，マクロ機能はプログラムのコーディングの効率を良くする

アセンブラの基本となる機能は，機械語の命令に1対1で対応するニモニックを機械語変換することです．しかし，実際のコーディングでは，同じような命令を繰り返し何回も記述したり，特定の処理のための命令のルーチンを，いたる所で書き込まなければならないことが生じます．

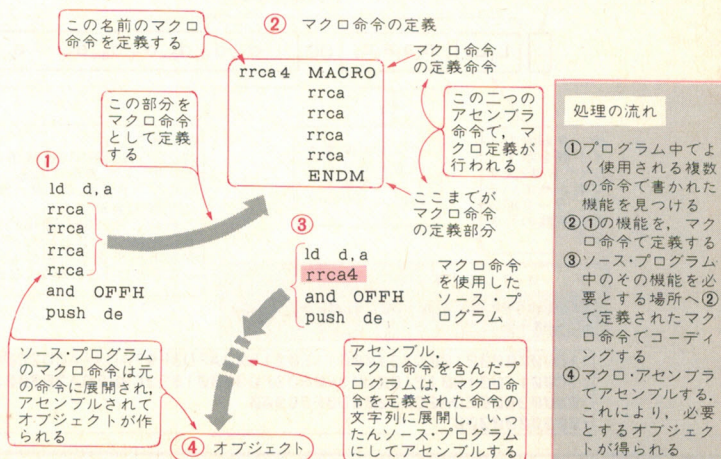
このような問題の解決のために，マクロ命令の機能がアセンブラに用意されています．Z80のアセンブラとして，各社からマクロ機能をもったものが発表されていますが，標準となるのは，マイクロソフト社のM80です．これは，最近価格も下がっていますので，入手も容易です．

このマクロ機能は，16ビット・マイクロコンピュータであるインテル社の8086のマクロ・アセンブラであるMASM（マイクロソフト社）とも多くの互換性をもっています．書きようによっては，マクロ機能をフルに利用することで，8ビット/16ビット共通な処理ルーチンを書くことも可能です．

マクロ機能とは，そんな発展性ももっています．

● マクロ機能で新しい独自の命令を作ることができる

マクロ機能とは，図9-1に示すようにプログラム中でよく使用される基本的な機能を，プログラマによって新しく定義し，命令として利用できる機能です．こ

〈図9-1〉マクロ命令の定義および用法

の新しく定義された命令は，ソース・プログラム中の任意の場所で利用でき，アセンブル時に，このマクロ命令はもとの命令群に展開されます．

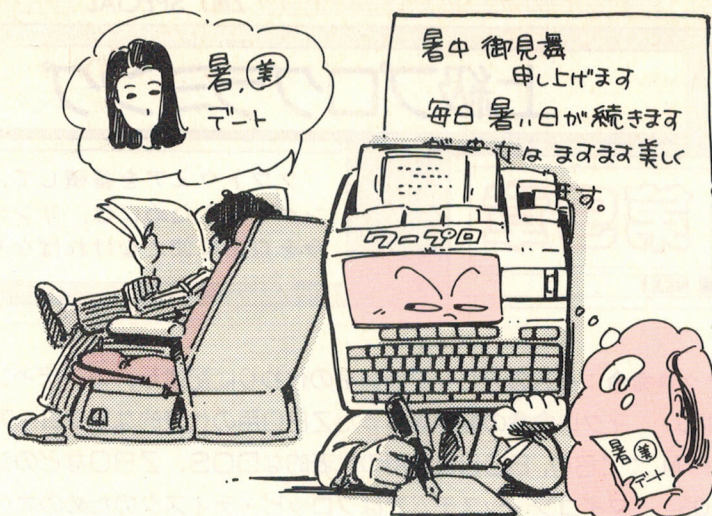
これは，例えばワープロにたとえると，熟語や短文を登録し，必要なところで簡単な読みを入力することで，正しい所定の文章が本文の中に展開されるようなものです．

ワープロと違うところは，ワープロではキー入力時に変換キーで変換されますが，アセンブラのマクロ命令は，その変換がアセンブル時に行われることです．したがって，ソース・プログラムには，その新しく定義されたマクロ命令が，そのまま記述されています．

〈図9-2〉 マクロ命令の使われ方

ワープロの辞書に相当する部分は，ソース・プログラム中にマクロ定義文として含めたり，辞書のように別ファイルとして用意することもできます．マクロ定義のファイルを読み込む命令も持っています．

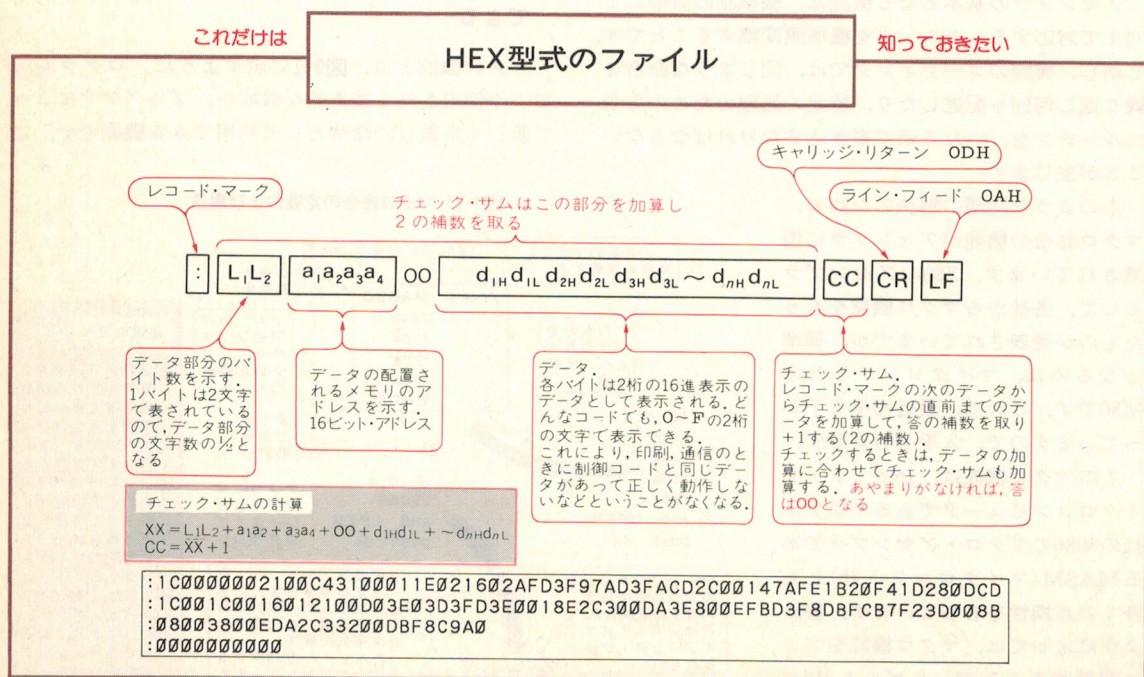
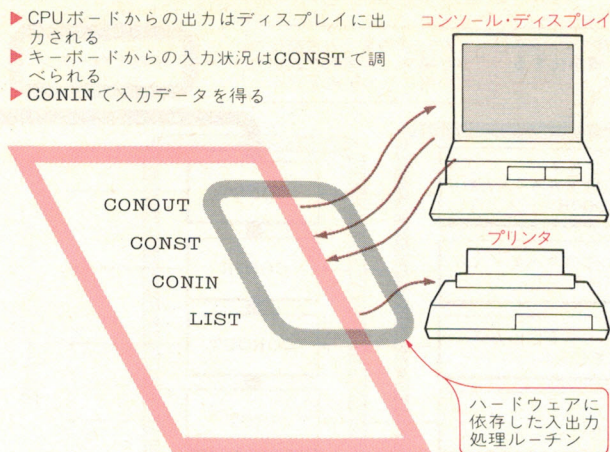
このようにマクロ機能を利用すると，独自の機能をもった命令を新しく作ることができます．この機能を中心に，命令体系を整備することができます．

使い込んでいくと，コンパイラのもっているようなレベルの命令を，アセンブラの中で自由に使用することもできます．のちほど，ファイルの読み込み手続きなど具体的な例で説明します．

この，機能を考えるということは，プログラムの開発という点で非常に重要なことです．このマクロ機能に注目できるかどうかが，システム・エンジニアとなるか，たんなるコーダで終わるかの差となります．

これを，機能という側面からみることと，その機能を実現するためのロジックを考えることとは別のことです．

マクロ命令によって新たに作成された命令を利用するときは，その命令の機能だけに注目して，その内部でのロジック処理内容に関知せずにすむのが最良です．そしてその命令の機能を実現するには，定められたデ

$$XX = L_1L_2 + a_1a_2 + a_3a_4 + OO + d_{1H}d_{1L} + \sim d_{nH}d_{nL}$$
$$CC = \overline{XX} + 1$$

```
:1CØØØØØØ21ØØC431ØØØ11EØ216Ø2AFD3F97AD3FACD2CØØ147AFE1B2ØF41D28ØDCD
:1CØØ1CØØ16Ø121ØØDØ3EØ3D3FD3EØØ18E2C3ØØDA3E8ØØEFBD3F8DBFCB7F23DØØ8B
:Ø8ØØ38ØØEDA2C332ØØDBF8C9AØ
:ØØØØØØØØØØ
```

〈図9-3〉基本となる入出力処理

上記以外にディスクとの処理，補助入出力装置との間の処理を用意する場合もある

ータの受け渡し部分からの情報のみで，その機能を実現するよう工夫しなければなりません．

効率を追及するあまりトリッキなプログラムになるよりも，明快なプログラムを作るよう心掛けるべきです．そのほうが寿命の長く，利用価値の高いプログラムとなります．

● 開発用とし標準となっているDOSであるCP/Mは，入出力処理をBIOSとして独立させている

利用の多いCP/Mでは，ハードウェアに依存するレベルでの入出力処理は，BIOS(Basic Input Output System)として独立しています．このBIOSの部分のみ，それぞれのハードウェアに合わせることで，容易にほかのシステムへ移植することができるように考えられています．

このように，プログラムを機能別に分割し，一部の仕様の変更がほかに影響しないようにします．プログラムの個々の機能を分割し，ほかの部分との関連を明確にしながらプログラムを作成するのが，プログラム作成の基本です．このような構造化プログラミングによって，プログラムは保守性が良く，汎用性の高いものに作りあげることができます．

最も基本となる具体的な入出力命令処理は，次のようなものです(図9-3参照)．

▶CONST

コンソールからの入力のステータスのチェック．

この処理では，入力の有無のチェックだけで，データの入力処理は行いません．

▶CONIN

コンソール・キーボードから入力データを得る．

多くの場合，入力の有無のチェックを行うルーチンも用意されています．ここでは，入力があるまで待ちます．

▶CONOUT

コンソール・ディスプレイへデータを出力する．

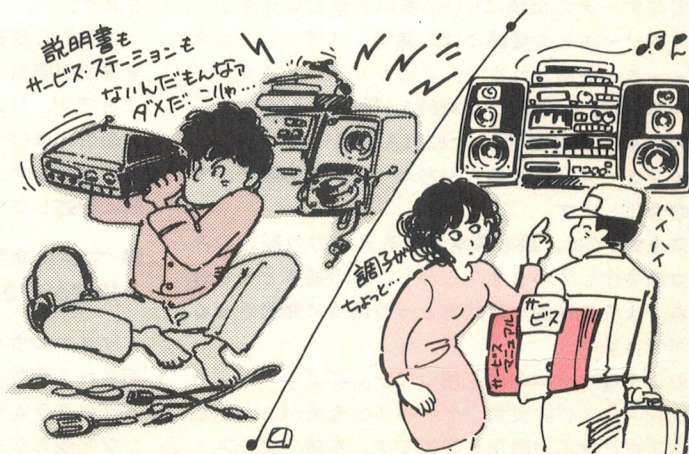
これだけは　知っておきたい

寿命の長いシステムはメインテナンスが配慮されている

コンピュータ・システムも，ほかの商品と同様に安心して使用するための条件として，トラブル発生時のメインテナンスの体制が整っている必要があります．

世の中の優れた商品では，故障などのトラブルが少ないことも確かですが，行き届いたアフタ・サービスの体制が整っていることも不可欠な要件です．

多くのパーソナル・コンピュータのユーザが作るプログラムは，プログラムの作成そのものに重点がおかれ，そのプログラムを使用する間に生じる各種のメインテナンスに対する考慮や，プログラム・ドキュメントも十分でないのが多くみられます．

プログラムのみ作って満足しているのでは，アマチュアといわれてもしかたありません．ドキュメント，メインテナンスの方法などのシステム設計の検討に，プロは7～8割を費やしています．

昔はテレタイプへプリントしていましたが，現在ではCRTディスプレイが普通です．

▶LIST

プリンタへデータを出力する．

これらの機能があれば，コンソールのキーボードからコマンドなどの指令を入力し，その結果をコンソールやプリンタに出力し，デバッグを行ったり，コンピュータの最小のシステムI/Oとして利用することができます(図9-4)．

もうひとつ重要なことは，この機能を利用する側からは，ハードウェアの違いにもかかわらず，いつも同じに見えることです．具体的にいうと，これらの機能を利用するときには，常に同じデータの受け渡し方法でデータの処理ができるため，プログラマはハードウェアの違いを意識する必要がないのです．

そしてこの基本となる処理ルーチンを使用して，図9-5に示すように順次，より複雑で限定された処理ルーチンが作られていきます．このように，階層化し各モジュールが独立して機能するようにプログラミングする手法が，大型機のプログラミング技術としても提唱されています．

次に具体的な例を示します．

〈図9-4〉 CP/MのBIOSに準じた入出力処理の例

〈図9-5〉 命令プログラムの機能を階層化する

入出力の基本となるマクロ命令を作る

シングル・ボードのコンピュータなどでも，コンソールとのデータの交換にI/O処理が必要となります．このコンソールとの接続には，通常シリアルのインターフェースが使用されます．このシリアル・インターフェースには，一般的には8251Aが利用されます．ここでは，第6章で説明した8251Aを用いた，シリアル・インターフェースの入出力ルーチンの処理を，マクロ命令化します．

リスト9-1では，最も基本となる入出力の部分のみマクロ命令化しています．しかし，この基本となる命令から，より具体的な目的をもった命令に発展させることができます．

このようなマクロ命令を用いてプログラミングすると，ハードウェアの変更，デバイスの変更があったときに，プログラムの修正が容易です．本体のソース・プログラムを変更することなく，入出力のマクロ命令の中身を変更し，再度アセンブルすれば，処理が終わります．

具体的な変更があったときに，プログラムを修正する場所が限定されることが，保守のうえからも重要なことです．リスト9-1では，デバイスの初期化のマクロ命令化は行っていません．実際のプログラムでは，この部分もマクロ命令として入出力の定義部と同じ場所に記述しておきます．

● マクロ命令の定義でもローカル変数，定数の指定ができる

マクロ命令を作成する場合，マクロ命令内でのみ使用される変数ラベルをどう扱うかが問題となります．

プログラムが大きくなると，変数や定数の管理が厄介な問題となります．とくに，プログラムをブロックごとに作成するときなど，ほかのブロックで使用した

〈リスト9-1〉入出力のマクロ化

```
                              .Z80
                         ;
0001                     txrdy     equ    001
0002                     rxrdy     equ    002
006E                     mode      equ    6eh
0037                     cmd       equ    37h
0044                     siod      equ    44h
0045                     sioc      equ    45h
                         ;
                         ;
                         siocmd    MACRO  cmd
                                   ld a,cmd
                                   out (sioc),a
                                   ENDM
                         ;
                         siostat   MACRO
                                   in a,(sioc)
                                   ENDM
                         ;
                         siood     MACRO  par1
                                   local  lp
                                   ld c,par1
                         lp:       siostat
                                   and txrdy
                                   jr z,lp
                                   ld a,c
                                   out (siod),a
                                   ENDM
                         ;
                         sioid     MACRO
                                   local  jpp
                                   in a,(sioc)
                                   and rxrdy
                                   jr z,jpp
                                   in a,(siod)
                         jpp:
                                   ENDM

0000'                              cseg
0000'                    istart::
0000'    F3                        di
0001'    3E 00                     ld a,00h
0003'    D3 45                     out (sioc),a
0005'    D3 45                     out (sioc),a
0007'    D3 45                     out (sioc),a
                                   siocmd 40h
0009'    3E 40            +        ld a,40h
000B'    D3 45            +        out (sioc),a
                                   siocmd mode
000D'    3E 6E            +        ld a,mode
000F'    D3 45            +        out (sioc),a
                                   siocmd cmd
0011'    3E 37            +        ld a,cmd
0013'    D3 45            +        out (sioc),a
                         ;
0015'    01 1020                   ld bc,1020h
0018'                    roop:     siood c
0018'    49               +        ld c,c
0019'                     + ..0000: siostat
0019'    DB 45            +        in a,(sioc)
001B'    E6 01            +        and txrdy
001D'    28 FA            +        jr z,..0000
001F'    79               +        ld a,c
0020'    D3 44            +        out (siod),a
0022'    0C                        inc c
0023'    10 F3                     djnz roop
                         ;
0025'    C3 0000                   jp 0000h
                         ;
                                   END istart
```

注釈:

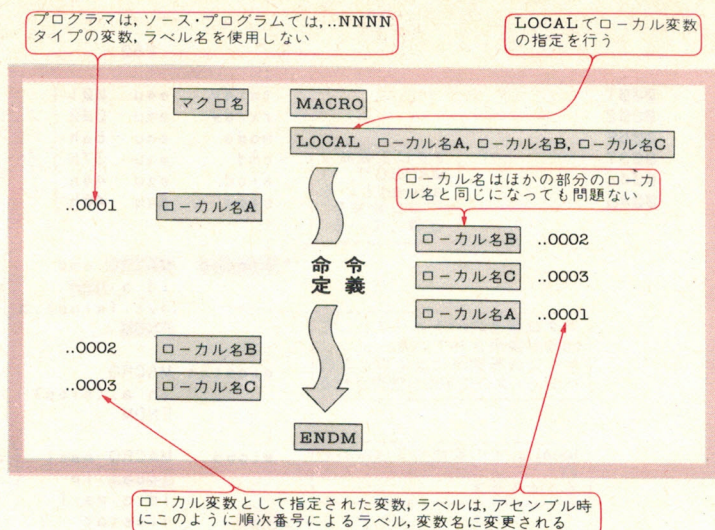
- プログラムの処理に必要なI/Oデバイスのアドレス,モード設定のためのコマンド・データなど,可能なかぎり定数としてソース・プログラムのわかりやすい場所に一括して定義しておく
- マクロ定義命令MACROで,siocmdというマクロ命令を定義している
- コマンドの設定をマクロ命令として定義する.パラメータcmdはコマンド・ワード
- マクロ命令のパラメータとしてcmdが使用されている.マクロの展開時にはそのときのパラメータの文字列が使用される
- マクロ命令の定義の終わりを示す
- ステータス・ポートを読み取るだけのマクロ定義もできる
- localは,その変数やラベルがそのマクロ定義内でのみ有効であることを示す
- siood 送信データ と記述して使用する.送信データとして,8ビット・レジスタ,数値が利用できる
- 送信可のチェックのループで,このマクロ命令内でしか利用しないのでローカル変数とする
- 送信可となるまでループする
- データ受信用のマクロ命令の定義.
- ステータスをチェックし,受信データがない場合,ZフラグをONにして次に進む.
- マクロ命令が展開されると,in a,(siod)の命令の次をjppが示す
- 受信データがある場合,Aレジスタにデータを読み込んで次に進む
- 以後コード・セグメントであることを示す
- この+は,マクロ命令より展開された命令コードであることを示す
- ソース・プログラムでは,この部分のみの記述でよい
- ..0000はローカル変数として定義された.これらの変数は..0000から..FFFFの順番のナンバがつけられる
- CP/M80のOSの制御にもどる
- プログラムのスタートのラベルをここに示す

変数についても考慮しなければならないとしたら，ブロック化する意味が半減してしまいます．

BASICの評判が悪いところのほとんどは，次のような欠点のためです．プログラムのブロック化を行うためにGOSUB命令を使っても，変数については常にプログラム全体について考慮しなければならないという欠陥です．

マクロ命令を使用する場合も，命令として利用する個々のマクロ命令内で定義された変数，ラベルを考慮しなければならないとしたら，とても大きなプログラムにはマクロ命令は利用できません．

図9-6に示すように，マクロ定義内で指定した変数，とくにラベルなどは，アセンブル時に個々の命令に展開されるとき重複しないように，M80は個々にローカルで定義されたラベル，変数に，順番に番号を割り当て重複して定義されることを防ぎます．

〈図9-6〉マクロ定義でのローカル変数の設定

● マクロ命令を定義した部分を独立したソース・ファイルとして作る

ソース・プログラムを作成するとき，毎回その先頭にマクロ定義部を記述するのでは，二重三重の手間となり，おもしろくありません．マクロ命令の定義部分のみ別ファイルとしてもち，管理することができれば，

これだけは

インターフェース・プログラムの作り方

知っておきたい

キーボード入力などのように，ハードウェアの制御を必要とする処理は，論理的なインターフェースとなるプログラムを作ります．このソフトウェアを介することで，ハードウェアの状況や仕様を意識することなく，プログラムが作れることになります．

この関係は図9-Aに示すように，ソフトウェアの立場からは論理的なデータの受け渡しだけの問題となります．ハードウェアが変わったとしても，この仲立ちのプログラムのみを変更すればすむので，移植性の高いものとなります．

またこの部分のプログラムを作る技術者は，ソフトウェアの細部について知る必要はなく，データの受け渡しの標準化された仕様にしたがってプログラムするだけですみます．

CP/Mなどの汎用のDOSでは，このような部分をBIOSとして独立したインターフェース・プログラムとしています．

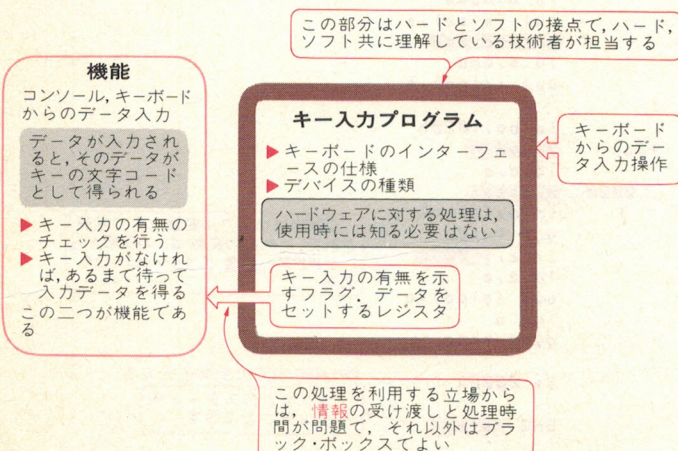
〈図9-A〉コンソールからのキー入力処理の例

この部分はハードとソフトの接点で，ハード，ソフト共に理解している技術者が担当する

機能
コンソール，キーボードからのデータ入力
データが入力されると，そのデータがキーの文字コードとして得られる
▶キー入力の有無のチェックを行う
▶キー入力がなければ，あるまで待って入力データを得る
この二つが機能である

キー入力プログラム
▶キーボードのインターフェースの仕様
▶デバイスの種類
ハードウェアに対する処理は，使用時には知る必要はない
キー入力の有無を示すフラグ，データをセットするレジスタ

キーボードからのデータ入力操作

この処理を利用する立場からは，情報の受け渡しと処理時間が問題で，それ以外はブラック・ボックスでよい

〈図9-7〉 INCLUDE命令により，ほかのソース・プログラムを読み込む

以後，アセンブラでプログラムを作成するときにあると便利な機能について，具体的に例をあげて説明していきます．個々の機能単位で説明していきますので，必ずしも統一がとれていません．各アプリケーション作成時の部品として考えてください．

● デバッグ時には実行中のCPUの状態の表示が必要

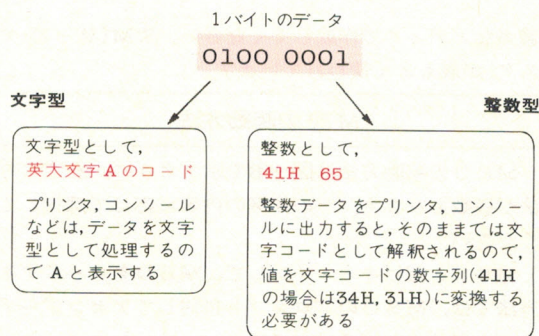
データはいくつかの型(タイプ)に分類できます．また区別しないと処理できないことが生じます．高級言語では，これらデータの型の区別が明確に指定されるのが普通になっています(図9-8)．

▶**文字型**：通常1バイトのデータで，このデータはそれぞれの処理系に応じた文字コードを示します．漢字処理の必要性から，文字型のデータも2バイト・コードで表す場合が多くなっています．

▶**整数型**：一般には16ビットのデータで，数値としての意味をもちます．2バイトで表現されるもの以外に1バイトのデータで数値としての意味をもたせて，バイト型と呼ばれるデータとして処理される場合もあります．

▶**実数型**：実数演算の対象となるデータで，それぞれの処理系に応じて型式が指定され，数バイト以上のデータ型式をもっています．

これらのデータは，それぞれのデータの型に応じた入出力の処理を必要とします．どの型の入出力であっても，コンソールとの間では文字型の処理が基本となります．したがって，これらの処理は各データの型の変換の問題となります．

〈図9-8〉 データの形
(同じデータでも，その解釈によって異なった姿を示す，どう解釈するかをデータの形として定義する)

そのマクロ命令のソース・ファイルを必要なときはいつでも，アプリケーション・プログラムの先頭で読み込んで利用できることになり，応用範囲が広がります．この機能も，M80はもっています．

include D：ファイル名 (Dはドライブ名)

の命令で，外部からソース・ファイルをその場所で読み込み，全体を一つのソース・ファイルとみなしてアセンブルしていきます(図9-7)．

この場合，ドライブ名，ファイル名は大文字で書いてください．小文字だと変数エラーとなります．

したがって，読み込まれるソース・ファイルは，必ずしもマクロ命令のソース・ファイルでなく，アセンブラで書かれたソース・プログラムであってもかまいません．共通に利用されるサブルーチン群なども，別ファイルとして作成しておくと便利です．

ハードウェアに依存した基本的な部分の処理をマクロ命令として定義し，これらをマクロ命令だけのソース・ファイルとして用意しておきます．具体的なアプリケーション・プログラムは，これらのマクロ命令を利用してコーディングします．このようにすると，ハードの仕様が変わったときの変更が容易となり，移植性の良いプログラムを書くことができます．

● レジスタなどの内容を16進(ヘキサ)表示するルーチンを作る

データは，型に応じた入出力処理を必要とします．しかしデバッグ時には，その型に無関係にデータの入出力を行うルーチンが必要となります．この場合には，各バイトごとに16進数の表示2桁のデータとして，型に無関係に処理を行います．

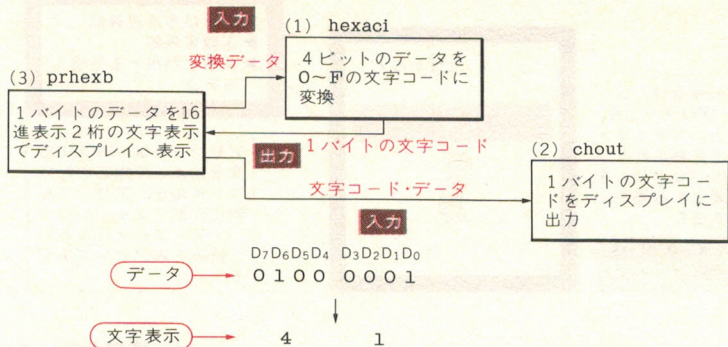
また，この機能を利用すると，CPUの各レジスタおよびメモリの内容のチェックが容易に行えます．これには，バイト・データを2桁の16進数データに変換するプログラムが必要となります．その具体的なプログラム例を示します．

まず，1バイトのデータを出力するプログラムを考えます．このプログラムの必要とする機能は，図9-9に示すような階層をもちます．

(1) 下位4ビットのデータを0～Fまでの対応する文字コードに変換する処理[1].

(2) 変換された文字コードをディスプレイに出力する処理.

(3) この二つの処理を制御して目的を達しようとする，今回対象となる上位の階層部分

〈図9-9〉 16進表示のプログラムの構造
(それぞれの処理も，より小さな機能の組み合わせで作られる)

アセンブラM80による アセンブルの方法

これだけは知っておきたい

ここでは，M80によるアセンブル作業の具体的な方法の説明をします.

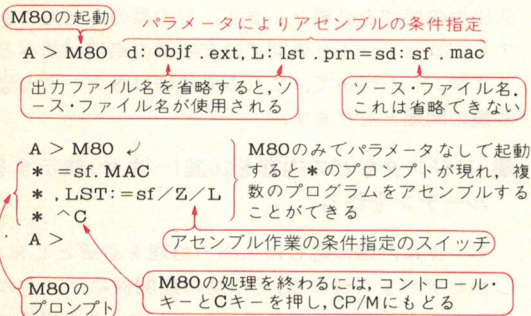
幾度か説明していますが，アセンブル作業は次の手順で行います.

(1) ソース・プログラムを作成する

(2) M80でリロケータブル・オブジェクトを作成する

(3) L80でリロケータブル・オブジェクトから実行型式のオブジェクトを作成する

基本的な作業は以上です. (1)の項目は，それぞれ適当なエディタで作成してください. WM(ワード・マスタ)が最も良く使われています.

〈図9-B〉 M80の使用法

	処理
O	リスト内のすべての番地などを8進数でプリントする(ALTAIR DOSでの標準)
H	リスト内のすべての番地などを16進数でプリントする(ALTAIR以外では標準)
R	強制的にオブジェクト・プログラム・ファイルを作成する
L	強制的にリスト・ファイルを作成する
C	強制的にクロス・リファレンス・ファイルを作成する
Z	Z80(Zilog型式)のニモニックで書かれたソース・プログラムをアセンブルする(Z80オペレーティング・システムでは標準)
I	8080Aのニモニックで書かれたソース・プログラムをアセンブルする(8080Aオペレーティング・システムでは標準)

M80の起動方法

M80の起動方法は図9-Bで示すように，設定の方法が複数あります. その作業の内容に応じて選択してください.

M80とだけキー・インして，M80をロードする方法では，M80のプロンプトに対してアセンブルするプログラムとパラメータをキー・インします. この方法は，複数のプログラムのアセンブルを続けて行う場合に，M80のロード時間を節約することができます.

M80の作業を終了するにはコントロールCで終了します.

単独のプログラムのアセンブルのときは，CP/M80のプロンプトに対してM80のコマンドとソース・プログラム名，アセンブル・スイッチの情報をパラメータとして与えます.

d オブジェクト・プログラムが書き出されるディスク・ドライブ名. 省略時はカレント・ドライブが選択される.

objf. ext
オブジェクト・ファイル名とエクステント名. 省略時はソース・ファイル名とRELのエクステントとなる.

L アセンブル・リスト・ファイルの出力される装置名. ディスク・ドライブ以外に，プリンタの場合はLST：，コンソールへ書き出す場合はCON：などとなる.

lstf. prn

これら三つのそれぞれの機能を，データの受け渡し部分では整合性をもたせ，ほかは独立性を確保しながらプログラムします．図9-10に16進表示に変換する処理のアルゴリズムを示します．Z80の命令の仕様，０～９，A～Fのコードのパターンを巧みに利用し，考えぬかれたプログラムとなっています．

図9-11には，１文字のデータを出力するマクロ命令の例を示します．このレベルのプログラムは，ハードウェアの仕様に依存する部分です．

CP/Mのシステム・コールの機能を利用する場合と，**リスト9-1**のシリアル・インターフェースを利用する場合の変更点を合わせて示しておきました．

ハードウェアの変更があったとしても，この部分のみの変更だけで対応でき，柔軟性のあるシステムが作れます．

図9-12には二つの機能を利用して，目的の16進表示を実現する方法を示しました．それぞれの機能をつなぎ合わせるだけで，所定の処理の目的がかなえられています．

下位のレベルとの間では，データの受け渡しの仕様だけで関連づけられ，処理の内容には立ち入っていません．出力処理の内容がハードの変更で変わったとしても，互いの独立性を保っています．

同様に，図9-13ではワード(16ビット)データの出力例を示します．個々のマクロ命令を利用するだけの簡単なものになっています．**リスト9-2**にこれらのソース・プログラムを示します．

ディスクへ書き出された場合のファイル名およびエクステント名．省略時はソース・ファイル名とｐｒｎのエクステントが付く．

ｓｄ　ソース・ファイルの入っているディスク・ドライブ名．省略時はカレント・ドライブが選択される．

ｓｆ．ｍａｃ

ソース・ファイル名とエクステント，ｍａｃのエクステントとなっている場合はエクステントを省略することができる．エクステントが省略されている場合Ｍ８０はｍａｃのエクステントとして処理する．

これらのコマンドの後にはスラッシュ／を書き，その後にアセンブラ・スイッチを設定することができます．このアセンブラ・スイッチによって，いくつかのアセンブラ条件をアセンブル時に指定することができます．

L80の起動方法

Ｌ８０の実行のためのコマンドは，図9-Cに示すようになります．この場合も，次の二つの方法でリンク作業が行えます．

Ｌ８０とのみキー・インしてリンカをロードします．その後，所定の各オブジェクト・モジュールを読み込むコマンドを与え，個々の作業を確認しながら進めます．最後にリンク・スイッチ／Ｅによって，作成されたオブジェクトをディスクに書き出し，Ｌ８０の実行を終了します．

〈図9-C〉 L80の使用法

●リンカの起動

```
A>L80
*DEU1:fname1.EXT／スイッチ,DEU1:fname2.EXT
*DEU3:fname3.EXT／スイッチ
*／E
```

リンクするファイルなどの条件をすべてパラメータとして入れる方式でもよい

リンカのプロンプト．実際にはこのプロンプトの間にリンカからのメッセージが表示される

リンクされたプログラムを，／Nで指定されたファイル名にCOMのエクステントを付けた名前でディスクに保存し，CP/Mのモニタへもどる

▶ DEU1,～DEU3は，それぞれのファイルの存在するディスクを示す．省略値はカレント・ディスク．

▶ EXTは，／Nスイッチ以外では省略値はRELとなる．

● SUBMITコマンドの使用

```
サブミット・ファイル　PRO.SUB
A:WM   D:$1.MAC                ;ワード・マスタでソースの修正
A:M80  D:$1,D:$1=D:$1／L／R;アセンブル
A:L80  D:$1,D:$1／N／E          ;リンク
D:$1                           ;プログラムの実行
                                ($1は処理時に入力されるパラメータ)
```

▶実行時

```
A>SUBMIT  PRO  Z80201
```

必ずドライブA

サブミット・ファイル名

パラメータ・プログラム名

自動的にそれぞれコマンドに従った処理が起動される

一方，これらのリンク作業のためのオブジェクト，リンク・スイッチの指定を，Ｌ８０のコマンドと同時にパラメータとして与えることもできます．

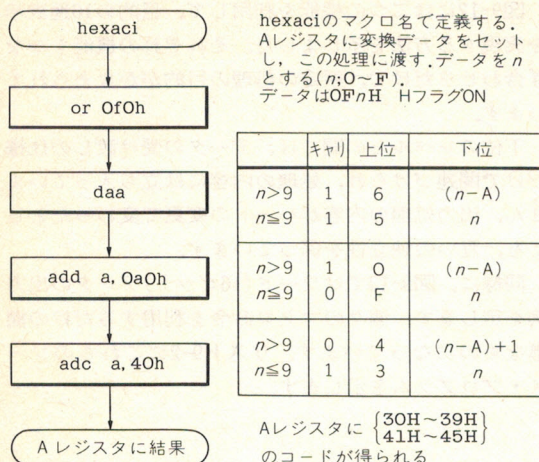
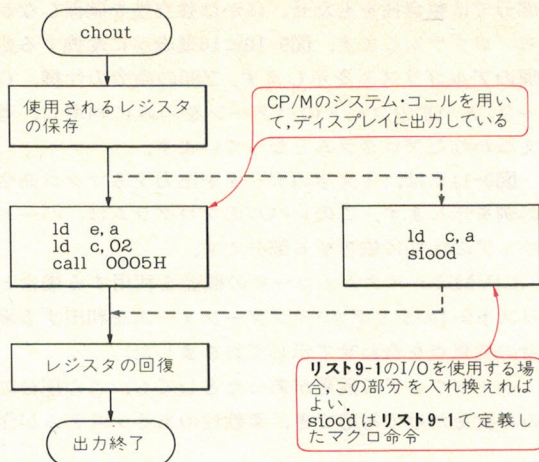
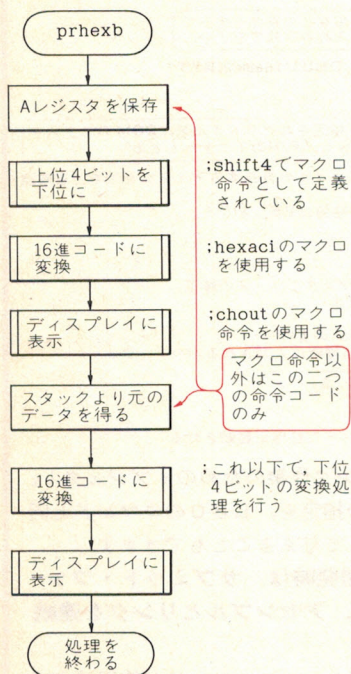
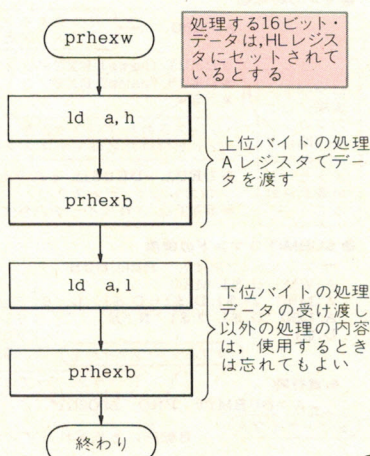
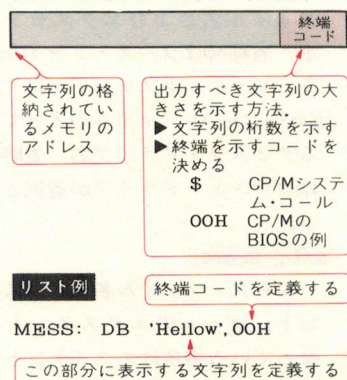
● プログラムの開発時は，サブミット・ファイルを作成して，アセンブルとリンクを連続して行う

CP/Mでは，サブミット・コマンドが使用できます．あらかじめ，アセンブルとその次に実行するリンクのコマンドおよびパラメータをセットしたファイルを作成しておくと，この機能を利用してそれらの作業を自動的に実行することができます．

この場合，アセンブルとリンクに先立って，エディタによるソース・プログラムの修正があるのが普通です．この作業を先頭に実行するようにしておきます．

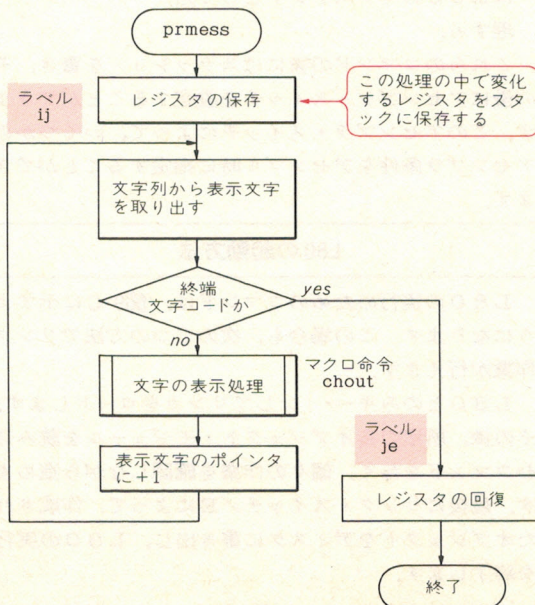
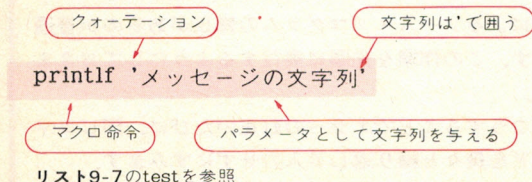
こうすることでエラーの訂正のたびに，同じコマンドを長々と繰り返して入力せずにすみます．

〈図9-10〉[4] 16進表示コードへの変換ルーチン

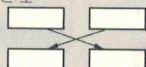
	キャリ	上位	下位
n>9	1	6	(n−A)
n≦9	1	5	n
n>9	1	0	(n−A)
n≦9	0	F	n
n>9	0	4	(n−A)+1
n≦9	1	3	n

〈図9-11〉 1文字出力のマクロ命令

〈図9-12〉 1バイト・データを16進表示で表示するマクロ命令の構造

〈図9-13〉 16ビット・データの16進表示ルーチン

〈図9-14〉 文字列の出力
(任意に設定された文字列の出力法)

〈図9-15〉 文字列の表示

〈図9-16〉 文字列表示のマクロ命令使用例

リスト9-7のtestを参照

〈リスト9-2〉マクロを用いた16進表示プログラム

```
        .Z80
;
savreg  MACRO
        push hl
        push de
        push bc
        ENDM
;
rstreg  MACRO
        pop bc
        pop de
        pop hl
        ENDM
;
sysc    MACRO  no
        ld c,no
        call 0005H
        ENDM
;
chout   MACRO a
        savreg
        ld e,a
        sysc 02h
        rstreg
        ENDM
;
; input data a reg.
hexaci  MACRO
        or 0F0h
        daa
        add a,0A0H
        adc a,40h
        ENDM
;
shift4  MACRO
        rrca
        rrca
        rrca
        rrca
        ENDM
;
; input data a reg.
prhexb  MACRO
        push af
        shift4
        hexaci
        chout a
        pop af
        hexaci
        chout a
        ENDM
;
; input data hl reg.
prhexw  MACRO
        ld a,h
        prhexb
        ld a,l
        prhexb
        ENDM
;
```


● 任意の文字列の表示を行うにはデータと命令を共に定義する

任意のメッセージを文字列として画面に表示し，操作の指示を行ったりする場合があります．この処理のため，１文字の出力をマクロ命令を使って，文字列の出力処理へと発展させます．文字列の出力のための処理は次のようなものになります．

> 指定された文字列を順次取り出して文字列の終わりをチェックし，終わりでなければ，その文字コードを文字出力マクロに渡す．以上の処理を文字列の終わりまで繰り返す．

この場合，制御するうえで次のような問題があります．つまり，文字列の終了をどのように決めるかということです．この方法には，次の二つが考えられます(図9-14参照)．

(a) 文字列の長さをデータとして受け渡しする．

(b) 文字列の終わりに，終わりを示すコードをセットしておく．この終わりのコードは出力しない．

(a)の文字列の長さを決める方法は，００Ｈ～ＦＦＨまでのすべてのコードを処理するような例では有効な方法です．文字の長さをアセンブラに計算させることもできますので，そう手間はかかりません．

(b)の文字列の終わりに終端を示すコードを置いておく方法は，わかりやすいのですが，その文字自身のコードを出力できないのが難点となります．

しかし，画面への出力であるのなら，キャラクタが割り当てられていないコードもあるので，問題は生じません．

CP/M80のシステム・コールでは，文字列の終わりに“＄”の文字を使用しています．この文字を画面に出力できなくなるので，ここの例では００Ｈを終端コードとします．

具体的なアルゴリズムを示すと，図9-15のようになります．この処理をコーディングしたのがリスト9-3です．この文字列表示のマクロ命令の使用法は，図9-16に示すように簡単なものです．

文字列の終わりで，改行するのかしないのかということも問題となります．改行・復帰のコードを，プログラマがその都度記入する方法もあります．しかし，改行付きのメッセージ表示，改行なしのメッセージ，改行・復帰のみ行う命令など簡単に用意できます．

● プログラムのソースは，命令記述，データ記述，絶対アドレスでの記述部分に分けられる

M80のソース・プログラムは，次の三つの性格をもった部分に分けて記述されます．

cseg

コード・セグメントと呼ばれる部分でプログラムの命令を記述する．

dseg

データ・セグメントと呼ばれ，データの定義を行う部分．

aseg

アブソリュート・セグメントと呼ばれ，絶対アドレスでアドレスの展開が行われる．これは，ソース・プログラムのorg命令で指定された絶対アドレスから順番に，命令コード，データ定義が割り当てられる．

csegとdsegの領域では，具体的なアドレスへの割り付けはリンカが行います．しかし，asegの領域では，プログラムの記述時点で絶対アドレスへの命令，データの設定が行えます．割り込みベクトル・テーブルの設定などで，特定のデータ・エリアを，絶対アドレスで指定するときなどに使用します．

ソース・プログラム中にこの3種類のセグメントが混在して記述されても，リンカによりcsegとdsegはそれぞれ順番にまとめられ，asegのデータと命令は，絶対アドレスに指定されます．

リスト9-3のように命令，データと交互に並んでいても，アセンブラはその中から命令部分とデータ部分をそれぞれ結合して，整理することができます(**図9-17**参照)．

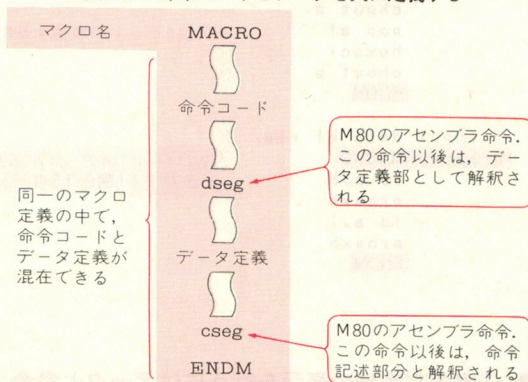
〈リスト9-3〉文字列の表示プログラム

```
;Z80903
;
;
;
prmess  MACRO  mess_ad
        local  ij,je
        push de    ;deレジスタのデータを保存する(ポインタに使用するため)
        ld de,mess_ad
ij:     ld a,(de)  ;メモリ中の文字列からデータを取り出す
        cp 0
        jr z,je
        chout a    ;一文字出力のマクロ命令を使用する．パラメータとしてはAレジスタの内容
        inc de     ;ポインタを進める
        jr ij
je:     pop de
        ENDM
;
prnt    MACRO  mess
        local pbuf
        prmess  pbuf
;
        dseg
pbuf:   db mess
        db 0
        cseg
        ENDM
;
printlf MACRO mess
        local pbuf1
        prmess pbuf1
;
        dseg
pbuf1:  db   mess
        db   0aH,0dH,00h
        cseg
        ENDM
;
pcrlf   MACRO
        chout 0ah
        chout 0dh
        ENDM
```


〈図9-17〉命令コードとデータを共に定義する

＊この機能を用い，図9-16の命令を実現する

● 16進表示のルーチンを用いてメモリの内容を表示する

データを16進表示するルーチンを利用して，メモリの内容を16進およびキャラクタとして表示するルーチンを作ります．メモリ中には，プログラムの命令コードとデータ部分があります．命令コードは，16進表示の部分で解釈します．データ部分は，キャラクタとして表示された部分で解釈します．

この機能は，DDTやZSIDのデバッガのメモリ・ダンプの機能と同様のものです．プログラムのアルゴリズムは図9-18に示すようになります．具体的なソース・プログラムを**リスト9-4**に示します．前もって定義されたマクロ命令を利用しているので，ソース・プログラムは簡単なものになっています．

このように，あらかじめ作成された各種の機能が自

〈リスト9-4〉メモリ・ダンプの処理プログラム

```
;Z80904
;
;
; input hl reg.                HLレジスタで示されるアドレスのメモリから，
;                              パラメータnで示されるバイト数だけ16進で
;                              表示する
dmpX     MACRO  n
         local  lp1            マクロ定義内のラベルをローカル変数とする
         push bc               BCレジスタの値をスタックに保存する
         ld b,n                表示バイト数をBレジスタにセットする
lp1:     ld a,(hl)             メモリより表示データを取り出す
         prhexb                マクロ命令，16進表示を行う
         inc hl                メモリ・アドレスのポインタを進める
         chout ' '             スペースを表示する
         djnz lp1              所定の回数繰り返す
         pop bc                BCレジスタを元にもどす
         ENDM
;
;
; input hl reg.                HLレジスタで示されるアドレスのメモリから，
;                              パラメータnで示されるバイト数だけ文字デ
;                              ータとして表示する
dmpc     MACRO  n
         local lp2,next        ローカル変数の指定
         push bc               BCレジスタの保存
         ld b,n                表示文字数をセット
lp2:     ld a,(hl)             メモリから表示データを取り出す
         cp 20h                コントロール・コードは"."に置き換えて表示する
         jr nc,next
         ld a,'.'
next:    chout a               データを画面に表示するマクロ命令
         inc hl                メモリ・アドレスのポインタを進める
         djnz lp2              所定の回数繰り返す
         pop bc                BCレジスタの値を回復する
         ENDM
;
;   HLレジスタで示されるメモリ・アドレスから16バイ
;   ト分，16進表示および文字タイプとして表示する
;
dmpm16   MACRO
         push hl               メモリ表示開始アドレスを保存
         dmpx 16               16バイト16進表示する
         prnt   '   '          スペースを表示する
         pop hl                メモリ表示開始アドレスを元にもどす
         dmpc 16               16バイト文字として表示する

         pcrlf                 改行を行うマクロ命令
         ENDM
;
;   表示開始アドレスを表示してから，メモリのダンプ
;   を行うマクロ命令
dmp16a   MACRO
         prhexw                表示開始アドレスを16進表示する
         chout ':'           } アドレスの表示部分とデータの表示部分を
         chout ' '           } :␣␣で分離する
         chout ' '           }
         dmpm16                メモリ・ダンプのマクロ命令
         ENDM
;
dumppg   MACRO  n              nで指定した行数分メモリのダンプを行う
         local lpp
         push bc

         pcrlf                 改行を行うマクロ命令
         ld c,n                表示行数をCレジスタへセット
lpp:     dmp16a                16バイト分の16進文字としての表示
         dec c                 カウンタを－1
         jp nz,lpp             カウンタが0になるまで繰り返す
         pcrlf                 改行する
         pop bc                BCレジスタを保存する
         ENDM
```

由に利用できるマクロ命令を利用して，プログラムの作成を行うと，効率よく進みます．

● CPUの状態表示ルーチンを考える

プログラムを作成するとき，テストをすると最初は思いどおりに動いてくれないのがほとんどです．とくに，アセンブラでプログラムを書いている場合は，絶えず思い違いやうっかりミスに悩まされます．そのような場合，プログラムにいくつかのチェック・ポイントをおき，プログラムの実行がその場所に来たときに，期待どおりの結果であるかどうかを調べるようにします．

ミスがあれば，いずれかの値が期待したものと異なっています．その値の違いから，ミスの原因が容易に推定できる場合がしばしばあります．

リスト9-5に示すマクロ命令は，プログラムの任意の場所にセットすることで，その地点でのCPUの各レジスタの値，フラグの状態を画面に表示します．

この**リスト9-5**では，プログラムの内容は表示するだけですので，表示の処理が終了したら次に進んでしまいます．画面へ表示する処理または表示ポイントが多数ある場合は，必要とする画面がスクロールして消えてしまう場合があります．

このため，所定のチェック・ポイントでプログラムの実行をストップさせます．中断されたプログラムは再実行できなければなりません．中断の方法としては，オペレータからの何らかのデータの入力を待つものとします．

中断させずにすべての表示をチェックする方法として，プリンタへの結果の表示を行う方法もあります．この場合，画面表示のマクロｃｈｏｕｔをｃｈｌｓｔなどという，プリンタへの１文字出力のマクロ命令に変えるだけですみます．

さらに，チェック・ポイントでプログラムが中断されたとき，CPUの状況だけでなく任意のメモリ・エ

リアの内容のチェックが必要となります．

次に示す，任意のアドレスからのメモリ・ダンプの処理を組み込むことで，より効果的なデバッグ処理が行えます．具体的な組み込み例は，次章で示すことにして，ここではメモリ・ダンプを行うルーチンを示します．

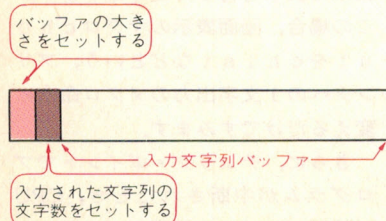
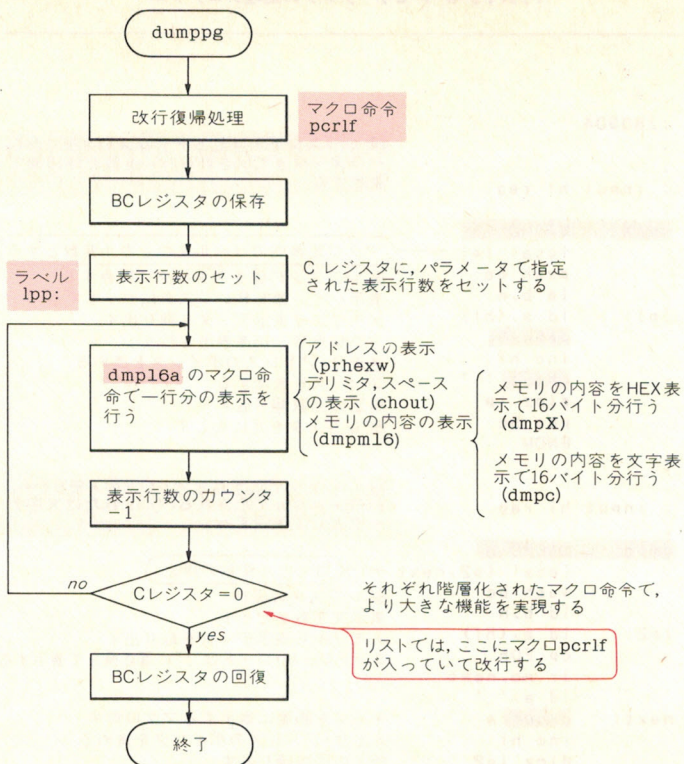
メモリのダンプは，任意のアドレスからの表示ができるようにします．任意のアドレスは，キーボードからオペレータが入力するものとします．このために，キーボードから入力された文字データを，16ビットのアドレス・データへ変換するルーチンが必要となります．

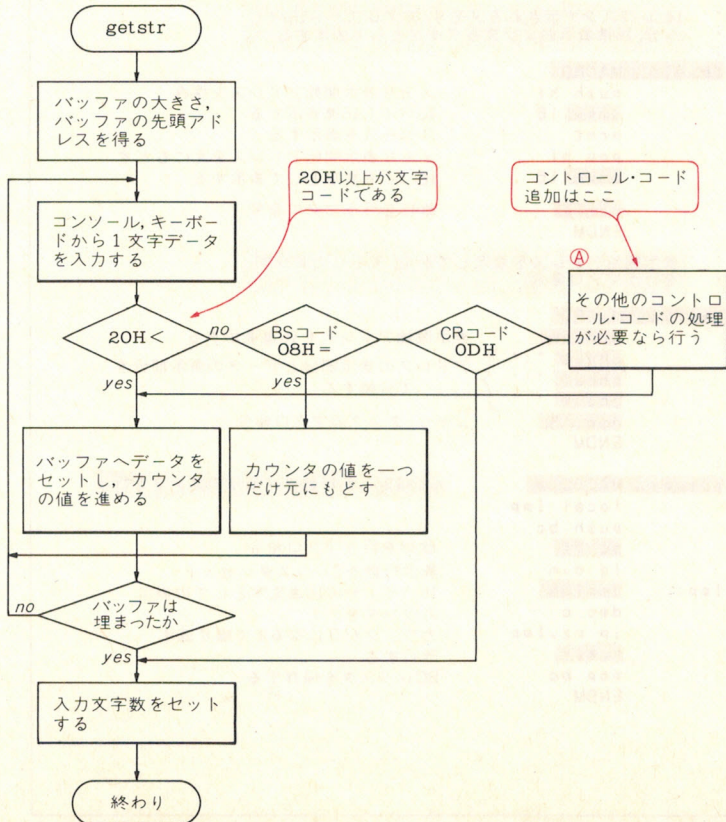
● 文字列を得る処理

バッファへ文字列を得る処理は，CP/MのDOSの機能にも用意されています．この機能を実現するプログラムを考えてみます．キーボードから入力された文字列は，256バイトの文字列バッファにセットされます．バッファの構造は図9-19に示すように，1バイト目にはバッファに格納する文字数の最大値，2バイト目には格納された入力文字列の文字数がセットされ，3バイト目から1バイト目で指定された文字数のバッファが続きます．

この文字数は1～FFまでの値です．入力データの修正は，バック・スペース・キーを使用して行います．その他のコントロール・キーの処理が必要だとしたら，図9-20のフローチャートのⒶの部分にその処理を自由に追加できます．

リスト9-6に，このプログラムの

〈図9-19〉 文字列入力バッファの構造

〈図9-18〉 メモリ・ダンプの処理

〈図9-20〉 文字列入力処理のフローチャート

〈リスト9-5〉CPUの状態を示すプログラム

```
;Z80905
;
        .Z80
        include B:Z80902.LIB
        include B:Z80903.LIB
        include B:Z80904.LIB
;       パラメータとしてどの位置でのデバッグ表示であるかを示す
        コメントを与える
debug   MACRO point
pophl   macro       マクロ命令内でマクロ定義を行うこともできる
        pop hl
        prhexw
        chout ' '
        endm
;
        push iy   この処理が終わったとき，すべて
        push ix   のレジスタを元の値にもどすた
        savreg    め，すべてのレジスタの値をスタ
        push af   ックへ保存する
;
        push iy   各レジスタの値をスタックに保存
        push ix   して順次取り出し，画面に表示する．
        savreg    そのため一時的，スタックに保存
        push af   する
        printlf point コメントを表示
        printlf ' af   bc   de   hl   ix   iy '
        pophl   AFレジスタの値を16進表示
        pophl   BCレジスタの値を16進表示
        pophl   DEレジスタの値を16進表示
        pophl   HLレジスタの値を16進表示
        pophl   IXレジスタの値を16進表示
        pophl   IYレジスタの値を16進表示
        pcrlf   改行
;                ←この部分に中断のマクロを入れ，改善が図れる
        pop af
        rstreg    すべてのレジスタ値を元に
        pop ix    もどして，元のプログラムに
        pop iy    もどる．
;
        ENDM
;
;
test::  ld a,41h
        ld hl,3132h
;
        debug 'debug test'
;
        jp 0000h
;
        end test
```

（図中注記：HL，DE，BCのレジスタを保存するマクロ命令 → savreg）

実行の過程および結果をデバッガでチェックした様子で示しています．

● 文字列を16ビットのアドレス・データに変換する

バッファに任意の文字列が入力できるとしたら，その文字列をアドレス・データとして解釈してメモリ・ダンプのスタート・アドレスとする処理を考えます．その処理を，リスト9-7にマクロ命令として定義します．

ｃｈｒｈｅｘは，DEレジスタの0～Fまでの文字型のデータを，バイトの16進数のデータに変換するマクロ命令です．入力データは，大文字を前提としています．そのために，小文字の場合大文字に変換するマクロｌｏｗｅｒを用意してあります．

ｌｏｗｅｒは，小文字データを大文字に変換するマクロ命令です．キーボードからの入力が大文字，小文字にかかわらず処理が行えるようになります．

〈リスト9-6(a)〉文字列出力のテスト・プログラム

```
;Z80906
;
        .Z80
        include B:Z80902.LIB
        include B:Z80903.LIB
        include B:Z80904.LIB
;
chin    MACRO       キーボードからの入力を行うマクロ命令．
        savreg      CP/M80のBDOSのシステム・コントロ
        sysc 01h    ールを利用する
        rstreg
        ENDM
;
;                   バッファの先頭アドレスがHLレジス
getstr  MACRO       タにセットされている
        local nn0,nn1,nn2
        savreg      レジスタの保存
        ld a,(hl)   バッファの先頭に入力バッファのリミットがある
        inc hl      バッファの次には入力文字数をセットする
        ld d,h      入力データのバッファへの格納のポインタ
        ld e,l      として，DEレジスタを使用する
        ld b,a      Bレジスタに入力バッファのリミットをセットする
        ld c,0      Cレジスタに入力文字数が得られる
        inc de      入力バッファのデータ入力域を示す
nn0:    chin        キーボードからの入力マクロ命令
        cp 1fh      コントロール・キー以外はnn1へ行く
        jr nc,nn1
        cp 08h      バック・スペース・キーのチェック
        jr nz,nn2   バック・スペース・キーでない場合，次に進む
        dec de      BSキーの場合のポインタ，入力文字数のカウンタ
        dec c       入力バッファのリミットのカウンタを一つもどす
        inc b
        jr nn0
nn2:    cp 0dh      CRキーのチェック
        ld b,1      CRキーの場合，処理を終わるので，リ
                    ミットを1にして次に進む
nn1:    ld (de),a 入力データをバッファへ格納する
        inc de
        inc c       ポインタ，カウンタをそれぞれ一つ進める
        dec b
        jp nz,nn0 b=0でなければ続ける
;
        ld (hl),c 入力文字数をセットする
        rstreg    レジスタを元にもどす
        ENDM
;
;
gstrbf  MACRO buff,n 文字列の入力バッファもマクロ
        ld hl,buff   命令内で定義する
        getstr
;
        dseg
buff::  db        n バッファの最大入力文字数
        db        0 入力文字数がセットされる
        ds        n 入力された文字列の格納される場所
        cseg
        ENDM
;
;
test::  gstrbf buff,4   4バイトのデータ入力の
;                       テスト・ルーチン
        jp 0000h
;
        end test
```

（図中注記：jr nc,nn1 ～ ld b,1 … コントロール・コードの入力処理部分）

これらを用いて4桁のアドレスをキーボードから得るマクロ命令が，ｇｅｔａｄｒです．

このプログラムを実行すると，内容を表示するメモリのアドレスの入力を要求するメッセージが，最初に表示されます．4桁の16進(HEX)表示でアドレスを入力すると，所定のエリアの内容を表示し，次の処理の指定を要求し停止します．

〈リスト9-6(b)〉 Z80906〔リスト9-6(a)〕のプログラムの実行を，デバッガZSIDを用いてチェックする

```
A>ZSID D:Z80906.COM
ZSID VERS 1.4
NEXT  PC  END
0180 0100 A9FF
#D100
0100: C3 09 01 04 00 00 00 00 00 21 03 01 E5 D5 C5 7E
      .  .  .  .  .  .  .  .  .  !  .  .  .  .  .  ~
0110: 23 54 5D 47 0E 00 13 E5 D5 C5 0E 01 CD 05 00 C1
      #  T  ]  G  .  .  .  .  .  .  .  .  .  .  .  .
0120: D1 E1 FE 1F 30 0D FE 08 20 05 1B 0D 04 18 E8 FE
      .  .  .  .  0  .  .  .  .  .  .  .  .  .  .  .
0130: 0D 06 01 12 13 0C 05 C2 17 01 71 C1 D1 E1 C3 00
      .  .  .  .  .  .  .  .  .  .  q  .  .  .  .  .
0140: 00 61 D0 FE 06 81 80 21 20 98 30 0A C2 A1 C0 07
      .  a  .  .  .  .  .  !  .  .  0  .  .  .  .  .
0150: 16 0B 45 C3 2E 00 00 02 00 25 C0 C0 12 D3 50 06
      .  .  E  .  .  .  .  .  .  %  .  .  .  .  P  .
#L
  0100  JP    0109
  0103  INC   B
  0104  NOP
  0105  NOP
  0106  NOP
  0107  NOP
  0108  NOP
  0109  LD    HL,0103
  010C  PUSH  HL
  010D  PUSH  DE
  010E  PUSH  BC
#P13E
#G
ABCD
01 PASS 013E
 -Z--- A=44 B=0000 D=0000 H=0103 S=0100 P=013E
 ----- A'00 B'0000 D'0000 H'0000 X=0000 Y=0000 JP   0000
*0000
#D100
0100: C3 09 01 04 04 41 42 43 44 21 03 01 E5 D5 C5 7E
      .  .  .  .  .  A  B  C  D  !  .  .  .  .  .  ~
0110: 23 54 5D 47 0E 00 13 E5 D5 C5 0E 01 CD 05 00 C1
      #  T  ]  G  .  .  .  .  .  .  .  .  .  .  .  .
0120: D1 E1 FE 1F 30 0D FE 08 20 05 1B 0D 04 18 E8 FE
      .  .  .  .  0  .  .  .  .  .  .  .  .  .  .  .
0130: 0D 06 01 12 13 0C 05 C2 17 01 71 C1 D1 E1 C3 00
      .  .  .  .  .  .  .  .  .  .  q  .  .  .  .  .
0140: 00 61 D0 FE 06 81 80 21 20 98 30 0A C2 A1 C0 07
      .  a  .  .  .  .  .  !  .  .  0  .  .  .  .  .
0150: 16 0B 45 C3 2E 00 00 02 00 25 C0 C0 12 D3 50 06
      .  .  E  .  .  .  .  .  .  %  .  .  .  .  P  .
#G

A>
```


分割してプログラムを作成する機能がM80に用意されている

プログラムの作成という作業は，いまだに手作業による一品生産で，工業化されていない部分です．その生産性の低さを改善するために，ソフトウェア・エンジニアリングの立場で，いろいろな提案が行われています．

その中の有効な手段の一つとして，プログラムのモジュール化があります．それぞれの機能ごとにプログラムを作成し，個々のプログラム・モジュールの機能を明確にして，その機能を利用できるようにします．

今までに述べたマクロ命令の使用も，モジュール化の一つです．

ここでもう一度，M80を用いてプログラムを作成する方法を振り返ってみます．

図9-21に示すように，ソース・プログラムからRELのエクステントの付いたリロケータブル・オブジェクトが作られます．このリロケータブル・オブジェクトは，プログラムのモジュール化を行い，その状況に

〈リスト9-7〉今まで作ったプログラムを元にしたメモリ・ダンプ・プログラム

```
; Z80907
        .z80

        include B:Z80902.LIB    リスト9-5, リスト9-6のマクロ定義部分を
        include B:Z80903.LIB    それぞれ別にファイルとしてエクステント
        include B:Z80904.LIB    をLIBファイルとして保存してある.
        include B:Z80905.LIB    アセンブル時にこれらのファイルを読み込
        include B:Z80906.LIB    み,プログラム中で使用されているものが展
                                開される
;
;
lower   MACRO                   英字の小文字を大文字に変換するマクロ命令
        local je
        cp 'a'                  a～zの文字コードのとき,AND命令によって
        jr c,je                 D5ビットを0にする.
        cp 7bh
        jr nc,je                   CP  Sで  A<Sのとき:キャリ  ON
        and 0dfH                            A≧Sのとき:キャリ  OFF
je:     ←                       ラベルをこのように付けることで,マクロ命令が展開されたとき,
        ENDM                    ラベルは次の命令を示す
;
;
;    ASCII  HEX                 0～9, A～Fの文字データで表された2桁16
; input data    DE              進(HEX)表示のデータを1バイトのデータに
; output data   A               変換する.文字データは大文字を前提とする
chrhex  MACRO
        local next,next2
        ld a,e                  下位データから変換する
        lower
        cp 3AH                  0～9までは下位4ビットのみ取り出す
        jr c,next               A～Fは文字コードが41H～46Hで,これに9を
        add a,09h               加算し,下位4ビットを取り出す
next:   and 0fh
        ld e,a                  変換結果をEレジスタに保存する

        ld a,d                  上位データを取り出す
        lower
        cp 3ah
        jr c,next2
        add a,09h
next2:  and 0fh
        shift4                  下位4ビットと上位4ビットを入れ替えるマクロ命令.
        add a,e                 保存してあった下位4ビットのデータを加算して,結果
        ENDM                    をAレジスタに得る

;
;
;
getadr  MACRO                   4バイトの文字データを入力バッファ
        gstrbf buff,4           buffへ得るマクロ命令
        push ix
        ld ix,buff              インデックス・レジスタへバッファの先頭アドレスbuff
                                をセットする
        ld d,(ix+2)             バッファ中の所定の位置のデータを取り出す
        ld e,(ix+3)             小文字の場合,大文字に変換するルーチンで処理し,
        chrhex                  DEレジスタへデータをセットし,文字データをバイト・
        ld h,a                  データに変換する
        ld d,(ix+4)             16ビット・データなので,4桁の16進文字表示となる.
        ld e,(ix+5)             したがって,再度バイト・データの変換を行う
        chrhex
        ld l,a
        pop ix
        ENDM

;
        cseg
test::  printlf ' key in address ' メッセージ表示
        getadr                  キーボードから4桁の16進表示のデータを入力し,HLレ
        dumppg 12               ジスタに結果を得る
;                               HLレジスタで示すアドレスから,1行16バイトとして12行
                                分のメモリの内容を表示する
        printlf ' next cr end ^C '
        chin                    メッセージを表示
        cp 03h                  キーボードからの入力が03Hコントロール・キーと
        jp z,0000h              Cキーのときは処理を終了
        pcrlf  ←改行のマクロ
        jp test                 繰り返す
;
        end test
;
```

(chin:キーボードからの入力のマクロ命令)

応じて必要なオブジェクト・プログラムを組み合わせて，要求された仕様を満足する，実行可能なプログラムを作成するために，重要な役割を果たしています．

● リロケータブル・オブジェクトは複数のオブジェクトを結合し 実行可能なオブジェクトを作る機能をもつ

モジュール化されたオブジェクトは，アセンブル時には，実行時にどのメモリ・エリアに配置されるかはわかりません．複数のモジュールと結合し，実行可能なプログラムを生成するには，ジャンプ先，サブルーチンとして定義されたそれぞれの処理ルーチンの呼び先が，プログラムの配置先の変動に応じて変化できるようにしなければなりません．

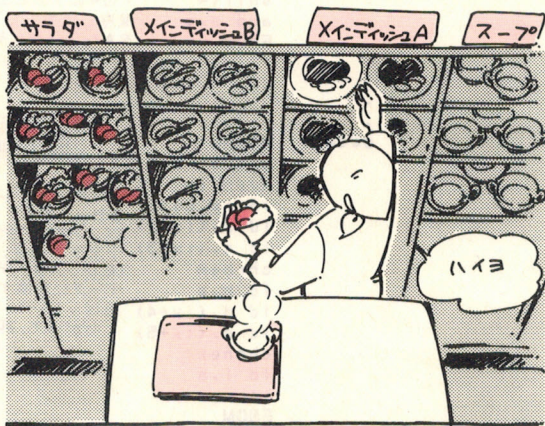
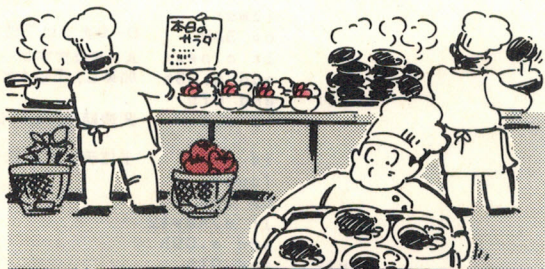
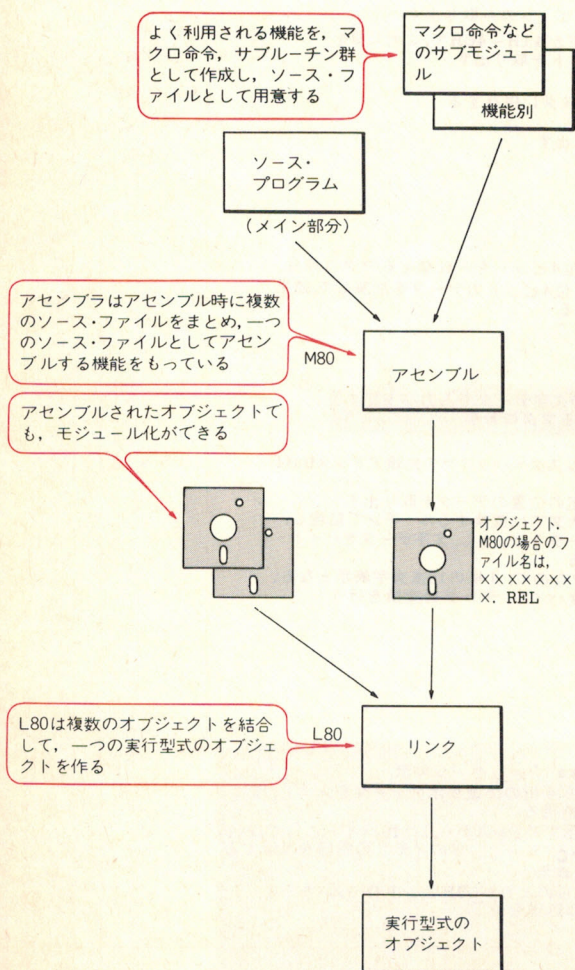
M80のアセンブル時には，各アセンブル・モジュールごとに，メモリへの割り付けは，次のように配置されます．アセンブルされたオブジェクトの先頭を０００Ｈ番地として，順番にアドレスがふられていきます．したがって，ジャンプ先などのアドレスは，そのモジュール内での相対アドレスで指定されます．

リンク時にリンカが，実際に配置されるメモリの絶対アドレスを計算します．

（モジュールの配置先の先頭の絶対アドレス）＋（モジュール内の相対アドレス）

と記述することで，モジュールの個々の命令コード配置先が，絶対アドレスで計算されていきます．

この処理で，各モジュールの命令のアドレスは決定できます．しかし，ほかのモジュールの命令やデータを参照する問題は解決されていません．その問題を解決する機能が用意されているので，次に示します（図9-22参照）．

〈リスト9-21〉モジュール化されたプログラムの開発手順

● 各オブジェクト・モジュール間での変数やラベルの参照方法

アセンブル時には，そのソース・プログラム中にある変数やラベルは，すべて参照可能です．しかし，同時にアセンブルされなかったオブジェクト間では，ラベルや変数などの参照はできません．

これらのオブジェクト・モジュール間の参照の最終処理は，リンカが行います．そして，ソース・プログラムの指定にしたがって，アセンブラは，それらのラベルや変数が外部のモジュールを参照するものであることを，またプログラム中のラベルや変数が外部のモジュールから参照可能であることを示します．また，そのためのアセンブラ疑似命令も用意されています(図9-23参照)．

リンカは，これらアセンブラによって示された情報に基づき，モジュール間の参照データのアドレスの割り振りを行います．

● 変数やラベルには大域的なものと，ローカルな有効範囲をもつものがある

これら複数のモジュール間で参照を可能とするため，ソース・プログラム中で，大域的(グローバル)な変数であるとの指定と，その変数がほかのモジュールで定義されていることを示す必要があります．

そのためのアセンブラ命令として，ＰＵＢＬＩＣおよびＥＸＴＲＮの命令が用意されています．

▶ＰＵＢＬＩＣ

このアセンブラ命令は，パラメータとして与えられたラベルまたは変数の名前が，このプログラム・モジュールと同時にロードされるほかのプログラム・モジュールからも，参照できることを示します．

▶ＥＸＴＲＮ

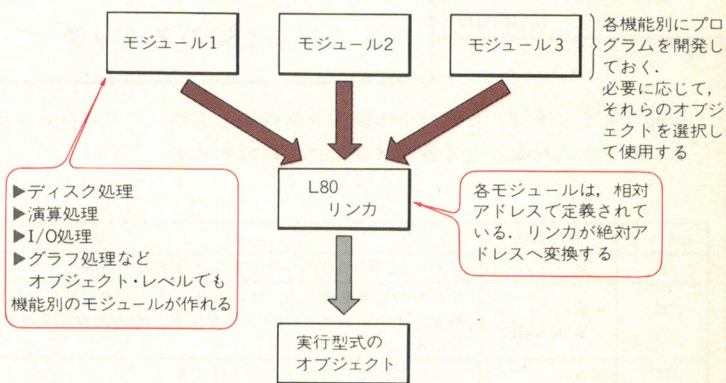
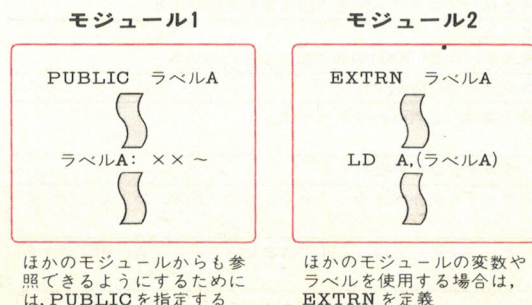
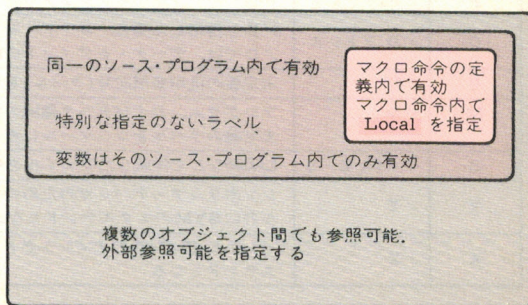
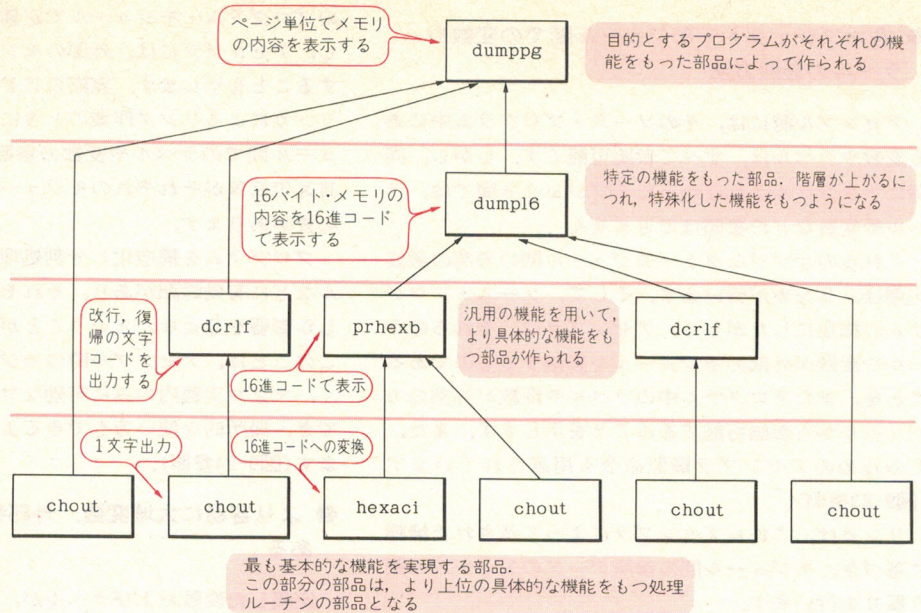
これは，そのプログラムで参照するラベルまたは変数の名前が，ほかのプログラム・モジュールで定義されていることを示し，アセンブラには，外部のモジュールのものを使用することを示します．実際のアドレスの割り当ては，リンカによるリンク作業のときに行われます．各モジュール間でのラベルや変数の参照は，ＰＵＢＬＩＣ，ＥＸＴＲＮがそれぞれのモジュールで定義されている必要があります．

プログラムを構造化し分割処理するためには，ラベルなどに有効範囲があり，それぞれのモジュール内にしか影響を与えないということが不可欠な機能です．このことは，アセンブル時のモジュールだけの話でなく，マクロ定義内のみに有効なローカル変数の指定もでき，局所的な使い方もできるように，配慮されています(図9-24参照)．

● より容易に大域変数，外部参照を示す方法がある

定義した変数およびラベルが，大域変数およびラベルであることを示す方法として，変数やラベルの後の：を：：と二つのコロンで示すこともできます．この方法だと，プログラムのコーディング中のラベルや変数を記述するときに，同時に定義でき便利です．

同様に，そのラベルや変数が外部参照であることを示す方法として，変数名の後に＃＃の記号を付加する方法もあります．

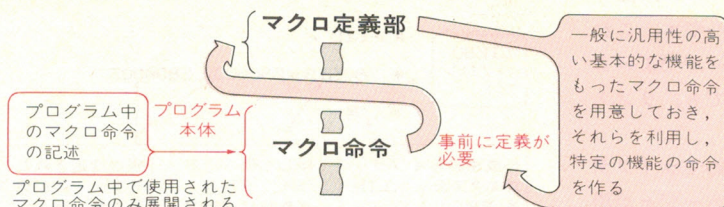
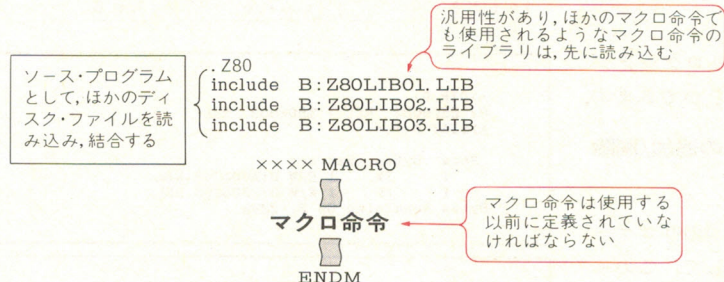
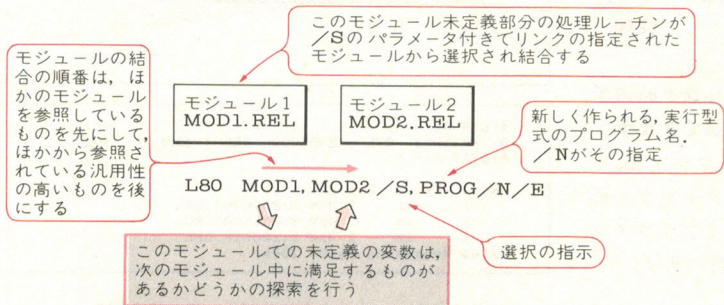
〈図9-22〉 モジュール化されたオブジェクトの結合

〈図9-23〉 ほかのモジュールの変数やラベルの使用するための定義

〈図9-24〉 変数，ラベル名の有効範囲

〈図9-25〉
プログラムの各機能を階層構造化する

階層化によるモジュール化

実際のアプリケーションで利用される数々の機能を，前もってプログラミングして，それぞれモジュール化しようとした場合，そのモジュールをどのように構成するかが問題となります．その一つの解決方法として，階層化構造によるモジュール化があります．

具体的なプログラムの各機能を，図化してみるとわかりやすいと思います．例として，メモリのダンプ・プログラム(リスト9-5のｄｕｍｐｐｇ)の機能構成図を図9-25に示します．

これだけは　**L80のスイッチ**　知っておきたい

Ｌ80のスイッチは，Ｌ80の全体動作を制御するものと，リンクの対象となる各オブジェクトに対するものがあります．

スイッチ	入力方法	処理内容
R	／R	リセット．処理を誤ったときなど今までの処理をリセットし，最初から処理をやりなおす
E	／E ／E:NAME	リンカの処理を終える．このとき，メモリ中のリンク処理を行ったオブジェクトをディスクに書き出し，新しいオブジェクト・ファイルを作る．NAMEが指定されている場合，プログラムの開始場所がNAMEとなる
G	／G ／G:NAME	リンクされたプログラムを実行する． NAMEの指定がある場合，そのアドレスより開始される
N	ファイル名／N	／E，／Gの実行時，このスイッチで指定されたオブジェクト・ファイル名としてディスクに保存される
P	／P:アドレス	リロケータブル・オブジェクトとして作成されたプログラム配置の開始番地を指定する
D	／D:アドレス	／Dと併用した場合，コード・セグメントのみ． ／Dでは，データ・セグメントの配置先を示す．ROM，RAM領域の指定などに用いる
U	／U	未定義の汎用参照記号(ラベル)をすべてコンソールに表示する
M	／M	プログラム，およびデータ領域の開始，終了番地，定義済みのラベル値． 未定義のラベルをコンソールに表示する
S	ファイル名／S	LIB80などで作成したファイルを指定し，参照することを指示する
Y	／Y	シンボリック・デバッガのためのラベルのファイルを作成する．ファイル名は／Nで，指定したファイル名にSYMのエクステンドとなる
X	／X	インテルHEX型式のオブジェクトを作成する．ファイル名は，／Nで指定したファイル名にHEXのエクステンドとなる

ここでは，最大四つの階層構造となっています．上位のレベルの機能を実現するためには，それより下位のレベルの各機能を必要とします．

上位のプログラムをアセンブルまたはリンクする場合，下位のレベルの処理ルーチンは完成されており，上位のプログラムによって読み込まれ，結合できるようになっていなければなりません．

また，ｃｈｏｕｔのように，どの機能からも参照されているようなものもあります．このように，広くいろいろな機能の処理プログラムから参照されるような基本となるプログラムは，汎用のライブラリにセットしておくと便利です．

また，仕事の目的に応じたライブラリも別に用意するようにします．そのためには，それらの階層の関係も明確にしておく必要があります．

● 各プログラム機能の階層状況に応じて，プログラムのリンクの順番も異なる

各ライブラリに含まれている機能の階層構造によって，プログラムの結合の順番が問題になります．

マクロ命令の定義部をまとめて，ライブラリとして別ファイルとすることができます．このソース・ファイルは，いつでもプログラム中に読み込むことができます．このライブラリを読み込む場所は，プログラムの最初もしくは利用される以前とします．

ソース・プログラム中で新たに定義するマクロ命令の場合も，同様に利用される以前に定義します．とくに，マクロ内で参照するマクロ命令は，注意しないとそのマクロ命令の定義される以前に定義するのを忘れ，エラーになります．

したがって，階層化されたライブラリを結合する場合，レベルの低い汎用の機能群のライブラリから，これらを利用して，より専用化されレベルの高い機能をもったライブラリの順番に読み込み，結合していきます．

リロケータブル・オブジェクトの結合の順番は，これとは逆になります．つまり，機能の専用化されたレベルの高いものを先にリンクします．そのとき，その機能が利用しているほかのモジュールのルーチンが未定義となってしまいます．その未定義の機能を，より汎用性の大きい下位レベルのモジュールの中から選択して，取り出し結合します．

こうすると，ライブラリ中の多数の機能のうち必要となる最小限の機能のみ，ロード・モジュール・オブジェクトとして結合されます．そのためコンパクトなオブジェクトを作ることができます〔図9-26(c)〕．

各ライブラリの機能を，必要の有無に関係なく結合するモードもリンカはもっています．このモードでリンク作業を行うと，結合の順番は任意でよくなります．しかし，できあがるロード・モジュールは，一般に大きくなります．

この現象は，ラン・タイム・プログラムとして，コンパイラのはき出だすオブジェクトに，よく見られます．

例えば，ＰＲＩＮＴ“Ａ”と書いた簡単なプログラムでは，アセンブラを用いたら，約10バイトでコーディングできます．しかし，BASICコンパイラのはき出すコードは，約数10Kバイトにもなるのです．これは，たんに文字を出力するためのモジュールでよいところを，三角関数などの演算ルーチンまでも，リンクしてしまっているためです．

リロケータブル・オブジェクトをまとめてライブラリを作り管理するため，ライブラリ・マネージャが用意されています．このライブラリ・マネージャはLIB80と呼ばれ，M80に付属しています．

このLIB80を用いて，各目的に応じたリロケータブル・オブジェクトのライブラリを作ることができます．

● LIB80はライブラリへのモジュールの追加/削除の管理機能をもっている

M80では，一度のアセンブル作業で一つのリロケータブル・オブジェクトが作られます．そして，このモジュールは，独立した一つのファイルとなります．順次開発したプログラムのモジュールが増えてくると，いくつかのモジュールを一つのファイルとして管理したくなります．

関連するモジュールを一つにすれば，リンク時の手間も少なくなり，リンクもれも防ぐことができます．

このアセンブラによって作成された1ファイル，1モジュールのリロケータブル・オブジェクトをまとめて，複数のモジュールを含んだ一つのライブラリ・ファイルを作ります．この処理はLIB80が行います．

この複数のモジュールを含んだファイルから，リンカは必要なモジュールのみ実行型式のオブジェクトへ選択することもできます．図9-27にこれらLIB80の機能を示します．

〈図9-27〉ライブラリ・マネージャLIB80の使い方

```
A>DIR B:
B: Z806002  REL : Z806003  REL
A>STAT B:*.*

 Recs  Bytes  Ext Acc
    1     1k    1 R/W B:Z806002.REL
    1     1k    1 R/W B:Z806003.REL
Bytes Remaining On B: 239k
```

```
A>LIB

*B:Z80LIB=B:Z806002,B:Z806003

*/E
```

```
A>DIR B:
B: Z806002  REL : Z806003  REL : Z80LIB  REL
A>STAT B:*.*

 Recs  Bytes  Ext Acc
    1     1k    1 R/W B:Z806002.REL
    1     1k    1 R/W B:Z806003.REL
    1     1k    1 R/W B:Z80LIB.REL
Bytes Remaining On B: 238k
```

● マクロ命令とコール命令を併用してオブジェクトを小さくする

リスト9-5で作成したマクロ命令は，マクロ命令のネスティング(入れ子)が多く行われていますので，ソース・プログラムでは，ステップ数が少なくても，展開されたオブジェクト・コードは大きなものとなります．

同一のマクロ命令を多数使用する場合，次の条件を前提としてコール命令を利用することができます(図9-28)．

〈図9-28〉使用頻度の高いマクロ命令はサブルーチン化を図る

＊上の説明は，パラメータの受け渡しがない場合の例．
パラメータの受け渡しがあるときは，その処理をマクロ命令内で行うようにする．

(1) マクロ命令の定義が少なくとも4バイト以上あること．

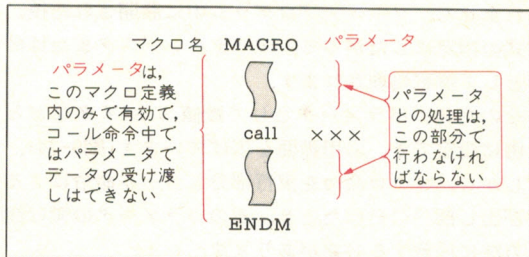
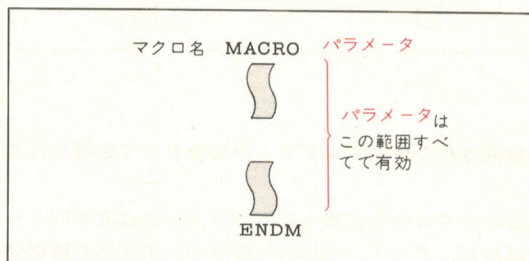
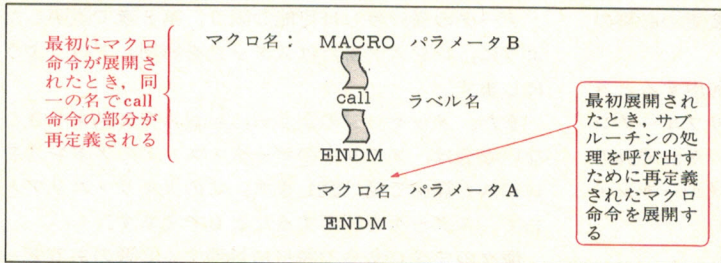
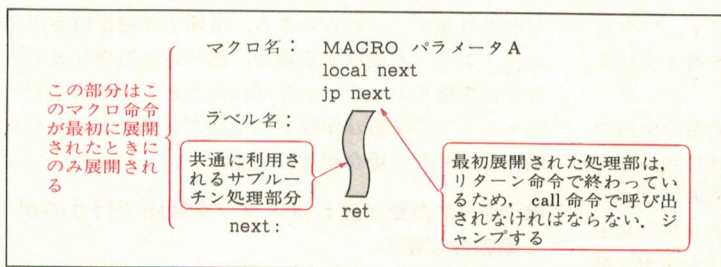
(2) コール命令を実行するための22ステートのオーバヘッドと，コール命令を呼び出した後，元にもどるためのRET命令(20ステート分)の処理時間のロス

〈図9-29〉 単純にサブルーチンが使用できる場合におけるメモリの節約の割合

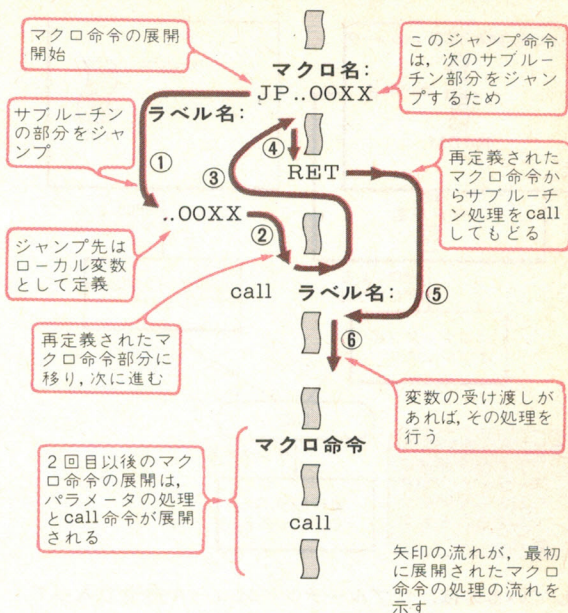
$$3\times n+(m+1)<n\times m$$

▶ nはプログラム中に現れるマクロ命令の頻度
▶ 3はコール命令のための必要なバイト数
▶ mはマクロ命令で展開される命令のバイト数

上記の関係を満足する場合，サブルーチンを使用する効果がある
右辺―左辺が節約されるメモリのバイト数

〈図9-30〉 マクロ命令のパラメータ処理

〈図9-31〉 サブルーチンを使用するマクロ定義(1)

〈図9-32〉 図9-31の処理の流れ

が問題にならないプログラムであること．

以上の条件を満たす場合，命令の実行部の展開は一箇所で行い，処理の必要となる部分にはコール命令をマクロ定義しておきます．処理スピードの点を除けば，全体をマクロ定義するのと同様に処理されます．

コール命令を使用することで節約されるメモリの容量は，図9-29にしたがって計算できます．

● サブルーチン処理をマクロ命令内で定義する方法

具体的に，サブルーチンを利用するマクロ命令を定義する方法を説明します．

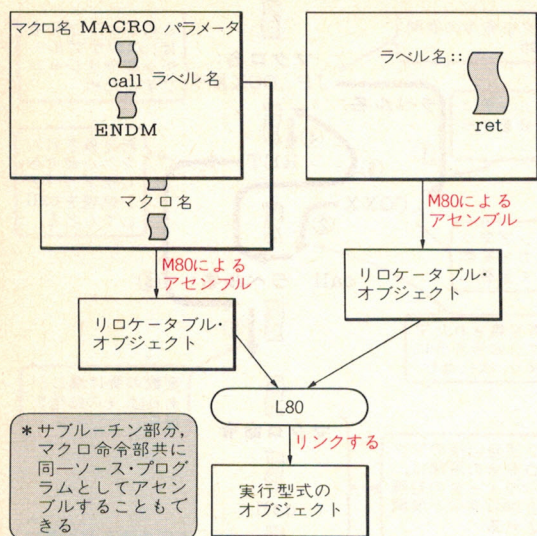
サブルーチン部分は，プログラム中に一箇所あればよいので，最初にマクロ命令が利用されたときにのみサブルーチン部分の展開が行われるようにします．二回目以降は，サブルーチンのコールのみ実行するようにします．

このためには，マクロ命令の再定義ができる機能を用いています．一度目のマクロ展開時に，サブルーチン部分と同じマクロ命令で，二回目以降に使用するマクロ定義を行います．新しく再定義されたマクロ命令は，サブルーチンをコールする部分だけになっています．

最初のマクロ展開時にも，サブルーチンをコールする必要があるので，再定義されたマクロ命令を使用しています．

このように，マクロ命令中にマクロ命令を使用することもできます．これらの様子を図9-30，図9-31，図9-32に示します．図9-31のサブルーチン部分をジャンプ

<〈図9-33〉サブルーチンを使用するマクロ定義(2)

〈図9-34〉マクロ命令のパラメータ

しているのは，サブルーチンへはコール命令で入っていかなければならないので，再定義されたマクロ命令部分へジャンプしているためです．

このプログラム中でのマクロ命令の再定義の手法は，その他にも多くの応用が考えられそうです．検討してみてください．

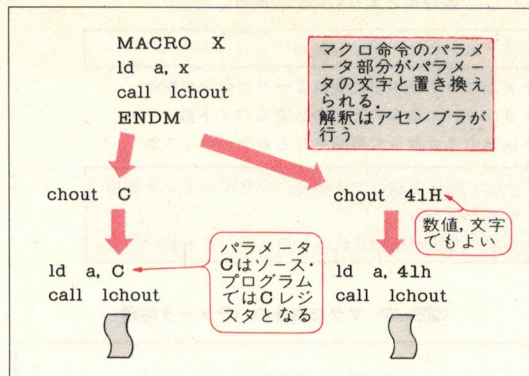
● コール命令部分とサブルーチン部分を別定義する方法もある

図9-33に示すように，サブルーチン部分とサブルーチンをコールするマクロ命令部分を，別にコーディングすることで，明解なプログラムとなります．マクロ命令の再定義などというややこしいことを考えずにすみます．

しかし，サブルーチン部分は，マクロ命令の使用の有無にかかわらず，すべて用意しなければなりません．また同一の機能を実現するために，二箇所のプログラムの管理を必要とすることは，メインテナンス上もトラブルの元となります．したがって，サブルーチン群は汎用性が高く，基本的な処理であまり変更の必要のないものを選びます．

この方法は，CP/MのBDOSの機能を利用するときなど，システム上，前もって用意されたサブルーチン群の利用などには最適な方法です．また，それぞれのシステムの基本的なI/O処理などを，サブルーチン・ライブラリとしてもちます．

● マクロ命令を使用する場合のパラメータの受け渡しの処理

マクロ命令では，マクロ命令に渡されたパラメータは，アセンブル時に解釈され，ソース・プログラム中に展開された後，プログラムの命令として処理されます．

このマクロ命令に渡されるパラメータは文字列として扱われ，データ，レジスタ，命令のいずれとも解釈はされません．ソース・プログラム中に展開された後，書式の指定にしたがって，データ，レジスタまたは命令として解釈処理されます．

そのため，パラメータとして数値，レジスタなどと自由に設定でき，応用範囲を広げています(図9-34)．

しかし，マクロ命令を実行部分とコール命令による呼び出し部分に分けたとき，そのパラメータの受け渡し方法に注意する必要があります．

コール命令によるデータの受け渡し方法には，レジスタ，スタックまたは特定の共通データ・エリア経由などがあります．これらのうち，全体で共通な特定のデータ・エリアを使用する場合，割り込み処理などで実行中に再度そのマクロが呼ばれたとき，その特定データ・エリアの内容が保証されず，プログラムが正しく動作しなくなる場合があります．

● データの受け渡しはスタック経由で行うのが柔軟性に富む

データの受け渡しは可能な限り，第8章で説明したように，レジスタまたはスタックを利用して行うようにします．

また，メッセージのように，レジスタに納まりきらない場合は，メモリ中のデータ・エリアのアドレスをレジスタ経由で受け渡します．このメモリ・エリアとして，スタックを利用することもできます．

個々のマクロ命令で独自に処理する必要のあるデータ・エリアの場合は，マクロ命令内でデータ・エリアを定義します．定義したデータ・エリア名は，ローカル変数とします．そうすれば，個々のデータ・エリアの独立性は保たれます．

〈リスト9-8(a)〉
マクロ化した1文字出力

1文字出力のマクロをサブルーチン利用のマクロに変更する

```
;
chout     MACRO x
          local next
          jp next
lchout::savreg
          ld e,a
          sysc 02h
          rstreg
          ret
next:
chout     MACRO x
          ld a,x
          call lchout
          ENDM
          chout x
          ENDM
;
hexaci    MACRO
          local next
          jr next
lhexac::or 0F0h
          daa
          add a,0A0H
          adc a,40h
          ret
next:
hexaci    MACRO
          call lhexac
          ENDM
```

- 展開されたサブルーチン部分をジャンプするため
- この部分が共通に呼び出されるサブルーチン部分となる
- マクロ命令内で再度同じマクロ命令を定義をし，以後はサブルーチンのコールを展開するようにする
- パラメータをAレジスタへ移動する処理は，マクロ定義部で行う
- 最初のマクロの展開時，サブルーチンを呼び出すため，再定義されたマクロ命令を使用する
- Aレジスタしか使用していないので，レジスタの保存マクロは使用しない
- ::はこのラベルがほかのモジュールからも参照可能なグローバル変数であることを示す．public lhexacの宣言と同一な効果がある
- パラメータの受け渡しがないのでコール命令のみ．Aレジスタ経由でデータが渡される

```
                              hexaci
                              ENDM
                          ;
0000'    3E 7A            test::  ld a,7ah
0002'    F5                       push af
                                  shift4
0003'    0F           +           rrca
0004'    0F           +           rrca
0005'    0F           +           rrca
0006'    0F           +           rrca
                                  hexaci
0007'    18 08        +           jr ..0000
0009'    F6 F0        +   lhexac::or 0F0h
000B'    27           +           daa
000C'    C6 A0        +           add a,0A0H
000E'    CE 40        +           adc a,40h
0010'    C9           +           ret
0011'                 +   ..0000:
                                  ENDM
0011'    CD 0009'     +           call lhexac
                                  chout a
0014'    C3 0024'     +           jp ..0001
0017'                 +   lchout::savreg
0017'    E5           +           push hl
0018'    D5           +           push de
0019'    C5           +           push bc
001A'    5F           +           ld e,a
001B'    0E 02        +           ld c,02h
001D'    CD 0005      +           call 0005H
0020'    C1           +           pop bc
0021'    D1           +           pop de
0022'    E1           +           pop hl
0023'    C9           +           ret
0024'                 +   ..0001:
                                  ENDM
0024'    7F           +           ld a,a
0025'    CD 0017'     +           call lchout
0028'    F1                       pop af
                                  hexaci
0029'    CD 0009'     +           call lhexac
                                  chout a
002C'    7F           +           ld a,a
002D'    CD 0017'     +           call lchout
0030'    C3 0000                  jp 0000h
```

- shift4のマクロの展開部
- サブルーチンをジャンプするローカル変数で定義した
- サブルーチンの部分
- hexaciの再定義されたマクロ命令
- データがAレジスタで受け渡されているので，HL，DE，BCのレジスタは，サブルーチン内でスタックへの保存ができる
- マクロsyscの展開部
- 2回目以後のマクロ命令の展開は，データの受け渡しとコール命令だけとなる
- CP/M80でのOSへもどる処理

● サブルーチン利用のマクロ展開例

リスト9-8は，1文字出力マクロ命令をサブルーチン化した例です．マクロ命令の定義部には，サブルーチン部分，マクロ命令の再定義などを行っている部分があります．

▶ｃｈｏｕｔは，マクロ命令の説明で使用していたものをそのまま使用しています．

▶ｘは出力データを表します．レジスタ,文字コードのいずれでもかまいません．

▶ｊｐ　ｎｅｘｔは，サブルーチン部分をスキップするためです．相対アドレスのジャンプにしなかったのは，何重にもマクロ命令がネストしている場合，128バイト以上になることが考えられるからです．ここでは，そんな大きくはならないのでｊｒ　ｎｅｘｔ　でもかまいません．

▶ｌｃｈｏｕｔは，サブルーチン部分の定義です．サブルーチンのラベル名は大域ラベルとなるように：：で定義してあります．

▶ｎｅｘｔで，サブルーチンの配置されたアドレスの次にジャンプしていきます．このｎｅｘｔのラベルは，このマクロ命令内でしか参照されないので，ローカル変数の定義をしてあります．

▶ｃｈｏｕｔで，このマクロ命令の再定義を行います．最初にマクロが展開されたとき，ここで再度自分自身を定義しなおします．

▶文字出力のため，再定義したマクロ命令を記述して処理を終わります．

マクロ命令とサブルーチンを組み合わせた，1文字出力のマクロ展開は上述のようになり，二回目以降の展開は，リストにあるように，データの受け渡しとサブルーチンのコール命令だけが展開されます．

● 外部サブルーチン・コールの例

リスト9-9～リスト9-12では，外部サブルーチン化されたライブラリの例を示します．

〈リスト9-8(b)〉実行テスト・プログラム

```
test::  ld a,7ah
        push af
        shift4
        hexaci      ← 最初の展開ではサブルーチン部分も展開される
        chout a     ←
        pop af
        hexaci    } この部分ではサブルーチン
        chout a   } 部分は展開されない

        jp 0000h
```

〈リスト9-9(a)〉文字出力マクロの外部サブルーチン化

```
                                 ;
0000'                     lchout::savreg
0000'     E5          +           push hl     }
0001'     D5          +           push de     } savregの展開部分
0002'     C5          +           push bc     }
0003'     5F                      ld e,a
                                  sysc 02h
0004'     0E 02       +           ld c,02h    } sysc 02Hの展開
0006'     CD 0005     +           call 0005H  } 部分
                                  rstreg
0009'     C1          +           pop bc      }
000A'     D1          +           pop de      } rstregの展開部分
000B'     E1          +           pop hl      }
000C'     C9                      ret
                                 ;
```

〈リスト9-9(b)〉リスト6-2(a)のアセンブル・リスト

```
;Z809009.MAC
;
        .Z80
;
        include D:Z8L9009.MAC  ← ほかのマクロ命令を使用しているので，マクロ命令のライブラリを読み込む
;
lchout::savreg
        ld e,a          このサブルーチンをモジュールとして
        sysc 02h        アセンブルする
        rstreg
        ret
;
        END
```

〈リスト9-10〉文字コードの変換ルーチンのモジュール化

```
0000'    F6 F0        lhexac::or 0F0h
0002'    27                   daa
0003'    C6 A0                add a,0A0H
0005'    CE 40                adc a,40h
0007'    C9                   ret

                              end
```

モジュール化されたプログラムの入力の名前は必ず大域(グローバル)変数にする

〈リスト9-11〉外部サブルーチンを呼び出すマクロ命令

```
chout   MACRO x
        ld a,x
        call lchout##
        ENDM
;
hexaci  MACRO
        call lhexac##
        ENDM
```

〈リスト9-12〉外部参照のプログラム例

```
; Z8Ø9Ø12.MAC          ソース・プログラム
;
        .Z8Ø           マクロ命令のライブラリのソース部分のプリントを,
        .XLIST    ←    行わないためのアセンブラ命令
        include D:Z8L9ØØ9.MAC
        include D:Z8L9Ø1Ø.MAC
        .LIST     ←    このアセンブラ命令以後はアセンブル・リストが作
;                      成される

test::  ld a,41h
        chout a        出力データとしてAレジスタの内容
        chout 'B'      出力データとして文字データ"B"が与えられる

        jp ØØØØh

        end test
------------------------------------------------------------
                                ; Z8Ø9Ø12.MAC
                                ;
                                        .Z8Ø
        この*のついたアドレスは,           .LIST
        外部参照アドレスであるこ   ;
        とを示す                            外部参照の指定

ØØØØ'   3E 41                   test::  ld a,41h
                                        chout a
ØØØ2'   7F              +               ld a,a
ØØØ3'   CD ØØØØ*        +               call lchout##
                                        chout 'B'
ØØØ6'   3E 42           +               ld a,'B'
ØØØ8'   CD ØØØØ*        +               call lchout##

ØØØB'   C3 ØØØØ                         jp ØØØØh

                                        end test
------------------------------------------------------------
Macros:
CHOUT   HEXACI  RSTREG  SAVREG  SHIFT4  SYSC

Symbols:
LCHOUT  ØØØ9*   TEST    ØØØØI'
        ↑ 外部参照アドレスを示す.リンク時にリンクが
          ほかのモジュールにあるlchoutのアドレスを割
          り当てる
No  Fatal error(s)
```

リスト9-9は，１文字出力の処理プログラムをサブルーチン化しています．ソース・プログラムとアセンブル・リストを示します．

リスト9-10は，ASCIIコードと16進(HEX)表示の変換処理を，サブルーチン化したものです．それぞれ別ファイルとしてアセンブルしたのは，後で，LIB80による複数のモジュールをもったライブラリ・ファイルのテストをするためです．

リスト9-11で，外部サブルーチンをコールするマクロ命令を定義します．このマクロ命令のみでライブラリ・ソース・ファイルを作っています．

リスト9-12で，これらのマクロ命令を用いたテスト・プログラムを作り，動作およびメモリ配置の様子を調べます．このプログラムを実行するには，外部サブルーチンを必要とします．そのため，**リスト9-9**のプログラムをリンクします．

● LIB80はリロケータブル・オブジェクト・モジュールの結合などのメインテナンスを行い，ライブラリを作る

このリンク時に選択/結合のテストを行うため，まず**リスト9-9**と**リスト9-10**のプログラム・モジュールをLIB80で結合し，複数のモジュールをもったライブラリ・ファイルを

作成します．これらのプログラムの関係は，図9-35に示すようになっています．

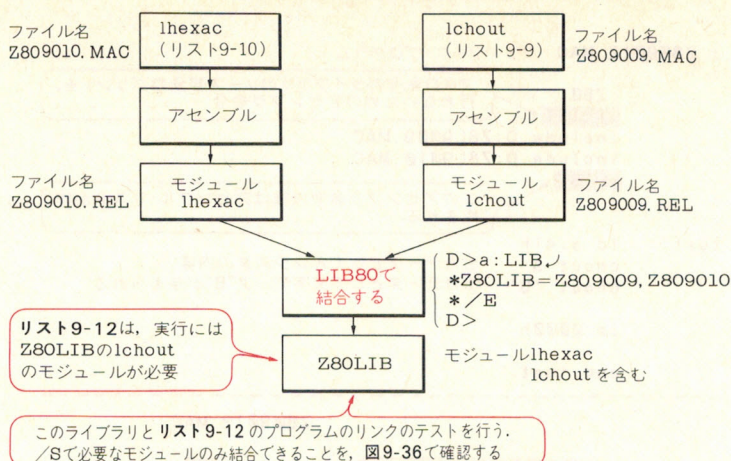
このライブラリ・ファイルと，**リスト9-12**をアセンブルしたリロケータブル・オブジェクトとで，リンク時のモジュールの選択/結合のテストを行います．

その様子を図9-36に示します．選択のパラメータ／Sを指定した場合と，しなかった場合のプログラムの大きさを比較しています．／Sを指定していない場合，**リスト9-12**で使用していないコードの変換ルーチンも含まれています．

本章では，プログラムを分割し構造化する方法の説明をしました．この方法は，汎用のコンピュータ・システムでプログラム作成の生産性を上げるために提唱されている方法のひとつです．

これらソフトウェアの生産性向上の努力は，ソフトウェア・エンジニアリングという一つの分野が形成されるまでになっています．現在，コンピュータ・システムに占めるソフトウェア費用は，ますます増大しています．今後も生産性向上の手法は多く発表されるでしょう．

ソフトウェアの作成は，コーディングのテクニック以上に保守も含めた生産性の向上のための考慮が必要になっています．

〈図9-35〉 LIB80でライブラリを作る

〈図9-36〉 ライブラリからのモジュールが選択される様子

Z80LIBより必要なモジュールのみ選択する／S

```
D>A:L80 Z809Ø12,Z80LIB/S,Z809Ø12S/N/E

Link-80  3.43  14-Apr-81  Copyright (c) 1981 Microsoft

Data     0103     011E     <     27>

40972 Bytes Free
[0103    011E        1]

D>A:ZSID Z809Ø12S.COM
ZSID VERS 1.4
NEXT  PC   END
0180 0100 A9FF
#D100,130
0100: C3 03 01 3E 41 7F CD 11 01 3E 42 CD 11 01 C3 00
      .  .  .  >  A  .  .  .  .  >  B  .  .  .  .  .
0110: 00 E5 D5 C5 5F 0E 02 CD 05 00 C1 D1 E1 C9 00 C1
      .  .  .  .  _  .  .  .  .  .  .  .  .  .  .  .
0120: 68 B8 59 31 D0 00 0C 98 86 90 9E AA A9 38 00 00
      h  .  Y  1  .  .  .  .  .  .  .  .  .  8  .  .
0130: 85
      .
#^C
D>
```

プログラムは0103H～011DHに配置された

Z80605Sはリンク作業によって作られるオブジェクト名．／N／Eでオブジェクトをディスクに保存してOSへもどる

ZSIDによって作られたオブジェクトの内容を確認する．必要なモジュールのみ結合されている

／Sの選択パラメータを指定しない

```
D>A:L80 Z809Ø12,Z80LIB,Z809Ø12/N/E

Link-80  3.43  14-Apr-81  Copyright (c) 1981 Microsoft

Data     0103     0126     <     35>

40955 Bytes Free
[0103    0126        1]

D>A:ZSID Z809Ø12.COM
ZSID VERS 1.4
NEXT  PC   END
0180 0100 A9FF
#D100,3130
0100: C3 03 01 3E 41 7F CD 11 01 3E 42 CD 11 01 C3 00
      .  .  .  >  A  .  .  .  .  >  B  .  .  .  .  .
0110: 00 E5 D5 C5 5F 0E 02 CD 05 00 C1 D1 E1 C9 F6 F0
      .  .  .  .  _  .  .  .  .  .  .  .  .  .  .  .
0120: 27 C6 A0 CE 40 C9 0C 98 86 90 9E AA A9 38 00 00
      '  .  .  .  @  .  .  .  .  .  .  .  .  8  .  .
0130: 85
      .
#^C
D>
```

011EH以後にコードの変換ルーチンが追加されている．Z80LIB全体が結合されている

これだけは　知っておきたい

Z80用デバッガZSIDの使用法

最後になりましたが，プログラムを完成させる途中では，デバッグの作業が長時間にわたることがあります．ZSIDは，Z80専用のデバッガで，よく使われるツールです．

ここでは，主に使われるコマンドの使用例を，トレースしながら示していきます(Ⓐから始まる)．

Ⓐ

CP/Mに属するデバッガにはDDTと呼ばれるものがある．しかし8080A用のために，Z80独自の命令の処理はできない．Z80の命令を処理するには，ZSIDを使用する．
コマンドはDDTとほとんどが共通である．

```
D>A:ZSID Z80904.COM
ZSID VERS 1.4
NEXT  PC  END
0180 0100 A9FF
#
```

- Z80904.COM ← ZSID(DDT)の処理の対象となるファイルの指定．コマンドのパラメータとして示す方法と，ZSIDのI，Rコマンドによる方法がある
- # ← ZSIDのプロンプト

```
D>A:ZSID
ZSID VERS 1.4
#-D100,180
0100: 01 F9 21 C3 3D 01 43 4F 50 59 52 49 47 48 54 20
0110: 28 43 29 20 31 39 37 37 2C 20 44 49 47 49 54 41
0170: E6 07 C2 7A 01 E3 7E 23 E3 6F 7D 17 6F D2 83 01
0180: 1A
```

- － ← －の表示がある場合，コードの下に文字としての表示は行わない
- D100 ← 表示開始アドレス
- 180 ← 表示終了アドレス
- Dコマンドは，指定されたスタートのメモリの内容をコンソールに表示する
- 0100:～0180: ← 表示アドレス
- メモリの内容の16進表示

ZSID(DDT)の終了は，^CまたはG0で終わる．

Ⓑへ続く

Ⓑ

```
#S100
0100 00 C3
0101 00 03
0102 00 01
0103 DB
0104 21 15
0105 CB .
#S10A
010A D3
010B 20 14
010C C9
010D DB
010E 21 15
010F CB .
#S113
0113 DB
0114 20 14
0115 C9
0116 31 .
#A116
0116 LD C,41
0118 CALL 103
011B CALL 10D
011E JP 0000
0121 LD BC,4243
0124 LD HL,4445
0127 NOP
0128 NOP
0129 .
#L116,128
  0116  LD   C,41
  0118  CALL 0103
  011B  CALL 010D
  011E  JP   0000
  0121  LD   BC,4243
  0124  LD   HL,4445
  0127  NOP
  0128  NOP
  0129
#P116
#P127
#P
01 0116
01 0127
#S100
0100 C3
0101 03 16
0102 01 .
#G100

01 PASS 0116
 ----- A=00 B=0000 D=0000 H=0000 S=0100 P=0116
 ----- A'00 B'0000 D'0000 H'0000 X=0000 Y=0000 LD   C,41
*0118
#T4
 ----- A=00 B=0041 D=0000 H=0000 S=0100 P=0118
 ----- A'00 B'0000 D'0000 H'0000 X=0000 Y=0000 CALL 0103
 ----- A=00 B=0041 D=0000 H=0000 S=00FE P=0103
 ----- A'00 B'0000 D'0000 H'0000 X=0000 Y=0000 IN   A,15
 ----- A=85 B=0041 D=0000 H=0000 S=00FE P=0105
 ----- A'00 B'0000 D'0000 H'0000 X=0000 Y=0000 BIT  0,A
 ----I A=85 B=0041 D=0000 H=0000 S=00FE P=0107
 ----- A'00 B'0000 D'0000 H'0000 X=0000 Y=0000 JR   Z,FA
*0109
#T
 ----I A=85 B=0041 D=0000 H=0000 S=00FE P=0109
 ----- A'00 B'0000 D'0000 H'0000 X=0000 Y=0000 LD   A,C
*010A
```

- #S100 ← Sコマンドは，指定されたアドレスから1バイトずつ順次指定された内容を書き込む．データの指定は16進表示で行う
- 0100～0102 ← ジャンプ命令を直接機械語で書き込む
- テスト機のI/Oデバイスのアドレスが異なっているので，Sコマンドで I/O命令のアドレス部分を書き換えている
- ピリオドは，セット・コマンドの終了を指示する
- #A116 ← Aコマンドはアセンブル命令で，指定されたアドレスが順次指定されたニモニックに対応する機械語でうめられていく
- CALL，JP ← コール命令，ジャンプ命令は，直接アドレスを指定する
- LD BC，LD HL ← メモリ中へは，セット命令で内容を書き換えられる．しかし，レジスタは直接書き換えられず，レジスタへの書き込みのルーチンを作り，その命令を実行し，レジスタの内容を変えてからデバッグすることができる
- #L116,128 ← Aコマンドの指定内容をLコマンドで確認する
- #P116 ← Pコマンドは，プログラムの実際を中断するポイントを設定する．ポイントは，アドレスとして指定する．このアドレスは，必ず命令の第1バイト目のアドレスでなければならない
- #P ← DDTの場合はPコマンドがないので，Gコマンドの2番目以降のパラメータとして，停止するアドレスを設定する．アドレスなしのPコマンドは設定されたパス・ポイントを示す
- #S100 ← 何度でも設定できる
- #G100 ← Gコマンドでは，指定されたアドレスから，プログラムを実時間で実行する
- *0118 ← パス・ポイントでは，プログラムは中断され，レジスタの内容などを表示し，次のコマンドを待つ
- #T4 ← Tコマンドは，次に示す数だけステップ動作で実行する
- IN A,15～JR Z,FA ← 入力データの情況の確認ができる
- #T ← ステップ数を指定しない場合，1ステップだけ実行して終わる

Ⓒへ続く

Ⓒ

```
#F100,130,00
#-D100,160
0100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0130: ~~ C5 11 2F 01 0E ~~ 53 20 31 2E 34 24 31 00 02
0150: 90 57 1E 00 D5 21 00 02 78 B1 ~~ ~~ ~~ ~~ ~~ 12
#M0,80,100
#D100
0100: C3 03 DA 00 03 C3 00 AA 16 02 AF D3 F9 7A D3 FA
      .  .  .  .  .  .  .  .  .  .  .  .  .  z  .  .
0110: CD 2C 00 14 7A FE 1B 20 F4 1D 28 0D 16 01 21 00
      .  ,  .  .  z  .  .     .  .  (  .  .  .  !  .
0140: 00 DA 80 04 19 00 02 ~~ ~~ ~~ 00 DA 3F 80 0F ~~
0150: 00 00 00 00 00 00 00 00 00 00 00 00 00 20 20 20
#IZ80904.COM
#R
NEXT  PC  END
0180 0100 A9FF
#D100,120
0100: 00 00 00 DB 21 CB 47 28 FA 79 D3 20 C9 DB 21 CB
      .  .  .  .  !  .  G  (  .  y  .     .  .  !  .
0110: 4F 28 FA DB 20 C9 31 03 25 B6 21 65 93 C5 0F A6
      O  (  .  .     .  1  .  %  .  !  e  .  .  .  .
0120: D8
      .
#L100,116
  0100  NOP
  0101  NOP
  0102  NOP
  0103  IN   A,21
  0105  BIT  0,A
  0107  JR   Z,FA
  0109  LD   A,C
  010A  OUT  20,A
```

- #F100,130,00 ← Fコマンドは，開始アドレスと終了アドレスで示されたメモリに，3番目のパラメータで示された内容を書き込む．ここでは，100H番地から130H番地まで00Hを書き込んだ
- #M0,80,100 ← Mコマンドは，メモリの内容の移動．ここでは，0番地から80H番地の内容を100Hから180Hまでに書き込む
- #D100 ← －のないDコマンドではメモリの内容に対応する文字コードを表示する．対応する文字がない場合は"．"ピリオドを表示する
- #IZ80904.COM ← Iコマンドでメモリに読み込むファイルを指定する
- #R ← Iコマンドで指定されたファイルをメモリに読み込む．カレント・ディスクからのみ読み込める
- #L100,116 ← Lコマンドは，指定されたアドレスから順次命令コードと解釈して対応するニモニックを表示する．つまり逆アセンブルを行う．データ部分などでは，データも命令コードとして解釈し，対応するニモニックを表示するので意味のない文字が表れる．Lコマンドで指定するアドレスは，命令コードの1バイト目を指定しないと正しい結果とならない

M80の疑似命令

疑似命令	記述形式	説明
●ニモニックの指定		
Z80	.Z80	Z80のニモニックを使用することを指示
8080	.8080	8080のニモニックを使用することを指示
●アセンブラの制御		
ASEG	ASEG	アドレスを絶対アドレスで設定する
CSEG	CSEG	命令コード領域のアドレスを相対アドレスで設定する
DSEG	DSEG	データ領域のアドレスを相対アドレスで設定する
COMMON	COMMON／name／	コモン領域のアドレスを相対アドレスで設定する
ORG	ORG exp	ロケーション・カウンタの値をexpに設定する
PHASE 〜 DEPHASE	.PHASE exp <program(s)> .DEPHASE	.PHASEから.DEPHASEの間のプログラムを，式expで指示した絶対番地からのプログラムとしてアセンブルすることを指定プログラムとしてアセンブルすることを指定
COMMENT	.COMMENT デリミタ text デリミタ	任意の区切り文字デリミタで囲んだテキスト文textをコメントとして指定する
INCLUDE $INCLUDE MACLIB	INCLUDE fname $INCLUDE fname MACLIB fname	INCLUDE, $INCLUDEおよびMACLIBは同じ意味をもつ．その記述位置に指定したインクルード・ファイルfnameを読み込み，展開することを指定する
RADIX	.RADIX exp	式expの値として2，8，10，または16を指定することで，基数表現を省略した場合の数値型式を指定する
END	END [exp]	プログラムの終了を指定する．式expは，必要に応じて実行開始番地を指定する
●シンボルの定義		
EQU	label EQU exp	式expの値を不変値として定義
SET ASET DEFL	label SET exp label ASET exp label DEFL exp	SET, ASET, およびDEFLは，同一の意味をもち，式expの値を可変値として定義する
●データの定義と領域の確保		
DB DEFB DEFM	[label] DB exp [,exp]… [label] DB string [,string]… [label] DEFB exp [,exp]… [label] DEFM string [,string]…	DB, DEFBおよびDEFMは，メモリ上にバイト定数を定義する．DEFBは式expの値を，DEFMはASCII文字コードstringの値を定義する．DBはどちらも定義可能
DC	[label] DC string	メモリ上に文字列定数を定義し，その最後の1バイトのMSBを1にする
DW DEFW	[label] DW exp [,exp]… [label] DEFW exp [,exp]…	DWおよびDEFWは，同一の意味をもつ．メモリ上に，ワード(2バイト)定数を定義する
DS DEFS	[label] DS exp [,value] [label] DEFS exp [,value]	DSおよびDEFSは，式expの値によって指示した大きさのメモリ領域を確保し，valueに指示した値で初期設定する
●リスト出力の制御		
PAGE $EJECT	PAGE [exp] $EJECT [exp]	PAGEおよび$EJECTは，改ページを指示する．式expの値によって以後の1ページの印字行数を指定する
TITLE	TITLE text	タイトルを指定する
SUBTTL $TITLE	SUBTTL text $TITLE ('text')	SUBTTLおよび$TITLEは同一の意味をもち，更新可能なサブタイトルを指定する
LIST	.LIST	アセンブリ・リストの出力指定
XLIST	.XLIST	アセンブリ・リストの出力の抑止を指定する
PRINTX	.PRINTX デリミタ text デリミタ	任意の区切り文字デリミタで囲んだテキスト文textをコンソールに出力する
LFCOND	.LFCOND	条件付きアセンブリで，偽条件となった本体のリスト出力指定
SFCOND	.SFCOND	偽条件となった本体のリスト出力の抑止指定
TFCOND	.TFCOND	偽条件となった本体のリスト出力/抑止の指定を反転する
LALL	.LALL	マクロ展開のリスト出力指定
XALL	.XALL	マクロ展開のコード生成文のリスト出力指定
SALL	.SALL	マクロ展開のリスト出力の抑止指定
CREF	.CREF	クロス・リファレンス情報の出力指定
XCREF	.XCREF	クロス・リファレンス情報の出力の抑止指定

M80の疑似命令

疑似命令	記述形式	説明
●リンケージ・エディタの制御		
NAME	NAME ('name')	モジュール名nameを指定する（先頭6文字まで有効）
REQUEST	.REQUEST fname [,fname]…	リンク時に参照するファイル名を指定する
EXT EXTRN EXTERNAL	EXT symbol [,symbol]… EXTRN symbol [,symbol]… EXTERNAL symbol [,symbol]…	EXT, EXTRN およびEXTERNALは，同一の意味をもつ．オペランドに指定したシンボルがほかのモジュールを参照（外部参照）することを定義
BYTE EXT BYTE EXTRN BYTE EXTERNAL	BYTE EXT symbol [,symbol]… BYTE EXTRN symbol [,symbol]… BYTE EXTERNAL symbol [,symbol]…	BYTE EXT, BYTE EXTRN, BYTE EXTENALは同一の意味をもつ．シンボルsymbolがバイトの値をもち，外部参照であることを定義
PUBLIC GLOBAL ENTRY	PUBLIC symbol [,symbol]… GLOBAL symbol [,symbol]… ENTRY symbol [,symbol]…	PUBLIC, GLOBAL, ENTRY は同一の意味をもつ．シンボルsymbolがほかのモジュールから参照されることを定義
●マクロ		
MACRO	name MACRO ダミー [,ダミー]… 〈macrobody〉 ENDM	本体macrobodyに記述したプログラムを，nameで指定したマクロ名ダミーでマクロ定義する
REPT	REPT exp 〈body〉 ENDM	本体bodyに記述したプログラムを，expで指定した回数だけ繰り返し展開する
IRP	IRP dummy, 〈parameters〉 〈body〉 ENDM	カンマで区切ったパラメータを順にdummyと置き換えながら，その文字数だけ本体bodyを繰り返し展開する
IRPC	IRPC dummy, characters 〈body〉 ENDM	指定した文字列を順に一文字ずつdummyと置き換えながら，その文字数だけ本体bodyを繰り返し展開する
ENDM	ENDM	マクロおよび繰り返し定義の終了指定
EXITM	EXITM	マクロ展開および繰り返しの打ち切り指定
LOCAL	LOCAL dummy[dummy]…	dummyによってマクロ内ラベルを指定する．記述は必ずMACRO疑似命令の次の行にする
●条件付きアセンブリ		
	IF ×××× | COND [argument] 〈body〉 [ELSE 〈body〉 ENDIF | ENDC	指定した条件××××が真のとき，以後の本体bodyをアセンブリする（そうでない場合は，以後の本体bodyをアセンブルする）
IF	IF exp	式exp の値≠0のとき 真
IFT	IFT exp	式 exp の値≠0のとき 真
COND	COND exp	式 exp の値≠0のとき 真
IFE	IFE exp	式 exp の値＝0のとき 真
IFF	IFF exp	式 exp の値＝0のとき 真
IF1	IF1	アセンブラが パス1のとき 真
IF2	IF2	アセンブラが パス2のとき 真
IFDEF	IFDEF symbol	symbolが定義ずみのとき 真
IFNDEF	IFNDEF symbol	symbolが未定義のとき 真
IFB	IFB 〈argument〉	ナル(null)引数のとき 真
IFNB	IFNB 〈argument〉	ナル(null)引数でないとき 真
IFIDN	IFIDN 〈arg 1〉, 〈arg 2〉	二つの引数(arg 1, arg 2) が同一のとき 真
IFDIF	IFDIF 〈arg 1〉, 〈arg 2〉	二つの引数(arg 1, arg 2) が異なるとき 真
ELSE	ELSE	条件の評価が偽のとき，以後に続くプログラムをアセンブルする
ENDIF	ENDIF	IF ××××で開始した条件アセンブリを終了指定する
ENDC	ENDC	CONDで開始した条件アセンブリを終了指定する

8048クロス・アセンブラ

■ NEXT

マクロ命令の実際の応用として，８ビット・ワンチップ・マイコンの代表的IC 8048のクロス・アセンブラの具体的作成法を示します．

keywords

イベント・カウンタ：8048は，１個のカウンタがあり，プロセッサに特別の負担をかけることなく外部の事象をカウントしたり，正確な時間遅延を生成するとき使用される．

ネスト：同一機能の命令，サブルーチンなどが，一つのプログラムの中に繰り返し組み込まれること．入れ子ともいう．たとえば，ＩＦ文の中に再びＩＦ文を組み込む場合など．

アーギュメント（引数）：マクロ定義で，マクロ命令（呼び出し側）からマクロ定義内部（呼び出される側）に情報を受け渡すのに使われる変数．

オブジェクト・ファイル：ソース・プログラムをアセンブラおよびコンパイラ（翻訳プログラム）で翻訳されて出力される目的ファイルのこと．

カレント・ページ：8048では，プログラム・メモリは2048語ごとにメモリ・バンク0，1に分割されている．さらにその中はページ（256語）に分割されている．その実行中の命令があるページのこと．

最近，小規模なシステムでもマイクロプロセッサが使われていますが，Z80や8085Aでは大きすぎる場合もあります．そんなとき8048シリーズ(8748, 8035)などの１チップ・マイコンを使用すれば，ハードウェアを小さくでき，経費も節約できます．そのような関係で，１チップ・マイコンの需要も増えています．

そこでアセンブラの問題ですが，CP/Mのもとで使えるクロス・アセンブラ(MAC48, XASM48)などのアセンブラも市販されていますが，ここでは一般的に知られているMACRO80のマクロ機能を使用した，8048のクロス・アセンブラを作りましたので解説します．

8048について

● 概要

8048は，インテル社の１チップ・マイコンで，いくつかのバージョンがあります．その一覧表を表10-1に示します．

8048は，次のような機能をもっています．

▶ 8ビット　CPU
▶ 1K/2K×8　ROM　プログラム・メモリ
▶ 64/128×8　RAM　データ・メモリ
▶ 8ビット×2　I/Oポート

〈表10-1〉MCS48ファミリ

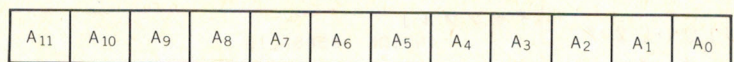
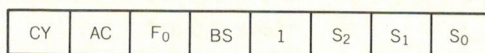
型　名	機　能	
8048	1K×8　マスクROM	64×8RAM
8748	1K×8　EP-ROM	64×8RAM
8035	外部ROM	64×8RAM
8049	2K×8　マスクROM	128×8RAM
8749	2K×8　EP-ROM	128×8RAM
8039	外部ROM	128×8RAM

▶ 8ビット×1　データ・バス
▶ 8ビット　タイマ/イベント・カウンタ
▶ 1レベルの外部割り込み
▶ 2本のテスト入力線
▶ クロック発振回路

その他，必要があればプログラム・メモリを最大4Kバイト，256バイトの外部データ・メモリを拡張でき，また8080系の周辺チップを簡単に接続できます．命令は，１ないし２マシン・サイクルで実行され，細く分類すると216の命令があり，約80%が１バイト命令です．したがってプログラム・メモリを効率良く使用でき，かつ高速性を合わせ持っています．

● アーキテクチャ

プログラム・カウンタは，図10-1のように12ビット構成され，最上位ビット(A_{12})はメモリ・バンク０，１を示し，SET MB×命令により選択されます．(A_{10}

〈図10-1〉 プログラム・カウンタ

〈図10-3〉
プログラム・ステータス・ワード(PSW)

〈図10-2〉 データ・メモリ・マップ

( )内の値は，8049/8749/8039，アドレスの値は，10進表示

$\sim A_8$)はページを示し，($A_7 \sim A_0$)はカレント・ページ・アドレスを示しています．プログラム・カウンタはリセット信号で“0”に初期設定されます．

プログラム・メモリは，マスクROM(8048/8049)，EP-ROM(8748/8749)，外付けROM(8035/8039)のバージョンがあります．プログラム・メモリには，次の三つの特別な番地があります．

▶ 0番地：リセット入力によって，0番地からの命令を実行する．

▶ 3番地：割り込み信号を受け付けると，3番地から始まるサブルーチンへジャンプする．

▶ 7番地：タイマ/カウンタのオーバフローによる割り込みを受け付けると，7番地から始まるサブルーチンへジャンプする．

プログラム・メモリは命令を格納するだけでなく，定数を格納することもできます．MOVP命令，およびMOVP 3命令を使用して，定数を参照します．MOVP命令は，MOVP命令のあるカレント・ページ・アドレスをアクセスし，そのデータをAレジスタに取り込みます．MOVP 3命令は無条件にページ3が参照され，3ページをデータ・テーブルとして活用できます．

内部データ・メモリ，および外部データ・メモリのアドレス指定は，R_0レジスタ，R_1レジスタのどちらかで，間接アドレス指定されます．外部データ・メモリはMOVX命令でアクセスします．

内部データ・メモリは図10-2のように構成されており，下記にその内容を示します．

▶ 0～7番地：レジスタ・バンク0のとき，0～7番地がワーキング・レジスタ($R_0 \sim R_7$)として直接アドレスすることが可能．

▶ 8～23番地：2バイトを1組として8レベルのスタックとして使用される．

▶24～31番地：SET RB 1命令でレジスタ・バンクとなり，24～31番地がワーキング・レジスタ($R_0 \sim R_7$)として使用される．例えば，バンク1を割り込み処理で使用し，バンク0をメイン・ルーチンで使用すれば，SET RB×命令で即座にレジスタの内容を退避，復活できる．

▶32～63(127)番地：汎用のRAMとして使用され，レジスタR_0，レジスタR_1により間接アドレスと指定される．また0～31番地を汎用RAMとして使用することもできる．

▶外部データ・メモリ：汎用のRAMで，MOVX命令により，レジスタR_0，レジスタR_1で間接アドレス指定される．

プログラム・ステータス・ワード(PSW)は図10-3のように8ビットのステータス・ワードであり，Aレジスタを通して内容の交換ができます．PSWの各ビットは，下記のように定義されています．

▶ビット0～2：スタック・ポインタ(S_0, S_1, S_2)を示す．

▶ビット3：未使用．

▶ビット4：ワーキング・レジスタ・バンクの状態を示す．
“0”＝バンク0，“1”＝バンク1

▶ビット5：ユーザによって制御されるフラグ0(F_0)

▶ビット6：補助キャリ(AC)を示し，ADD命令で発生し，10進補正命令DAAで用いられる．

▶ビット7：キャリ・フラグ(CY)を示し，前の演算によって，アキュムレータにオーバフローが発生しているかどうかを示す．

● 命令セット

8048は上記のように，すべての命令が1ないし2バイトで，その約80%が1バイト命令になっています．また8080Aの論理演算では，アキュムレータに対してのみ使用できますが，8048の場合I/Oポートも対象になります．従ってコントローラの応用に必要な，I/Oポートに対するビット単位の処理を1命令で実行できます．

```
「サンプル」
        ORI      P1,01H  ;PORT1 BIT0 SET
        ANI      P1,0FEH ;PORT1 BIT0 RESET
```

このように，8048は大変効率よくプログラム・メモリを使用できます．8048は，データ転送命令，演算命令，フラグ操作命令，分岐命令，タイマ命令，マシン制御命令，I/O命令などがあります．その詳細はSAMPLEプログラムかマニュアルを参照してください．

● マクロ・アセンブラとは

マクロ・アセンブラが通常のアセンブラ(例えば，CP/Mに付属しているASMなど)と違うところは，マクロ定義機能をもっていることです．マクロ定義とは，次のようなものです．あらかじめCPUの命令と疑似命令を使用して処理の内容を記述しておきます．そして，定義したマクロに対して名前を付けておくと，マクロに付けた名前をアセンブラ・プログラムの新しい命令として書くことができます．従って，いろいろなマクロを定義しておくことにより，プログラミングを容易にすることができるわけです．

例として，REPEAT-UNTILなどのプログラム制御機能をマクロ定義すれば，構造化プログラミングが可能となり，大変見やすいソース・プログラムを書くことができます．極端な話しですが，ソース・プログラム＝ドキュメントとすることも可能です．また1チップ・マイコン(8048など)の命令をマクロ定義すれば，クロス・アセンブラを作ることも可能です．

MACRO80について

MACRO80はマイクロソフト社のアセンブラで，強力なマクロ・アセンブラです．マクロ機能のほかに，いろいろな疑似命令をもっています．

● 疑似命令

疑似命令とはマイクロプロセッサではなくアセンブラに対する指示です．ここで今回のクロス・アセンブラで使用した疑似命令の一部をまとめてみます．

▶INCLUDE

INCLUDEステートメントの位置に，ほかのファイルを挿入します．

```
「サンプル」
        INCLDE  8048.LIB
```

クロス・アセンブラ用のマクロを定義し，一つのファイルとしてディスクにセーブしておき，ソース・プログラムの先頭にＩＮＣＬＵＤＥ ８０４８．ＬＩＢとすることにより，MACRO80が8048クロス・アセンブラとなります．

▶SET/DEFL/ASET

アーギュメントで指定した〈式〉の値に〈シンボル〉を割り当てます．

```
「サンプル」
        FLG      SET      0
```

SET命令は，EQU命令と違い再定義することができます．従って，アセンブル時の制御フラグとして使用することもできます．例えば，

```
「サンプル」
        ERR      DEFL     0
        ERROR    MACRO    X
                 PRINTX   <X>
      -------------------------
      : ERR      DEFL     ERR+1 :
      -------------------------
                 ENDM
```

このように，ERRORマクロが呼び出されるごとに，ERRをインクリメントします．従ってマクロ内部で発生したエラーの数を知ることができます．

▶EQU

アーギュメントで指定した〈式〉の値に〈シンボル〉を割り当てます．

```
「サンプル」
        A        EQU      07H
        R0       EQU      08H
        R1       EQU      09H
```

EQU命令は一度定義したシンボルに対し二度と定義することはできません．再定義する場合は，SET命令を使用します．

▶PRINTX

アセンブル中，〈テキスト〉をコンソールに表示します．

```
「サンプル」
        IF2
          PRINTX * INVARID OPERAND *
        ENDIF
```

マクロ内部で発生したエラーの内容をコンソールへ表示するとき役立ちます．．ＰＲＩＮＴＸはパス１およびパス２の両パスでテキストをコンソールへ表示しますので，上記のようにすれば，パス２のときのみ出力され見やすくなります．

▶DB/DEFB/DEFM

アーギュメントで指定したストリングスまたは式の値をもち，1バイトずつのデータ領域を確保し，初期値をセットします．

```
「サンプル」
        RETR     MACRO
               ----------------
               : DB       93H  :
               ----------------
                 ENDM
```

マクロ定義を使用したクロス・アセンブラで出力されるオブジェクト・ファイルは，すべてDB命令によります．

▶ASEG

CPモードを絶対アドレス・モードにします．

```
「サンプル」
                 ASEG
                 ORG       0H
```

ASEG命令はORG命令と共に使用され，ロケーション・カウンタを実アドレスにセットします．このクロス・アセンブラは，絶対アドレス・モードでのみ使用できます．

● 条件付きアセンブラ

条件疑似命令はアセンブラ時に特定の条件を調べ，その結果に応じてアセンブルします．

▶IF××××−ENDIF

条件が真の場合には，IFからENDIFの間にある命令をアセンブルします．また偽の場合には，IFからENDIFの間にある命令が無視されます．

```
「サンプル」
        IF1
        INCLUDE 8048.LIB
        ENDIF
```

この例では，アセンブラがパス1を実行中の場合のみ真となり，パス1のとき8048マクロ・ライブラリを挿入します．

▶IF××××−ELSE−ENDIF

条件が真の場合には，IFからELSEの間にある命令をアセンブルします．また偽の場合は，ELSEからENDIFの間にある命令をアセンブルします．

```
「サンプル」
        ADI      MACRO    X,Z
      ----------------------------------------
      :   IFIDN    <X>,<A>                   :
      :   DB       03H,Z                     :
      :   ELSE                               :
      :   ERROR    <INVARID OPERAND>         :
      :   ENDIF                              :
      ----------------------------------------
          ENDIF
```

変数Xが“A”のとき真となりオブジェクトを出力し，偽のときエラー・メッセージをコンソールに表示します．

● マクロ定義について

プログラマがマクロを使用する場合，あらかじめマクロ定義しておかなければなりません．マクロ定義は，次の書式で書きます．また，ダミー・パラメータをもたないマクロもあります．マクロ定義の中で別のマクロを呼び出し，展開することもできます．マクロのネストの深さはメモリ容量で決まります．

「書式」

```
  マクロ名  MACRO  (ダミー，ダミー)
              :
              :
            ENDM
```

```
「サンプル」
        RET       MACRO
                  DB       83H
                  ENDM
```

上記の例ではマクロ名にRETが使用されています．ここに注目するとRETはZ80の命令としてアセンブラ内部で使用されています．従って，このような使い方をすると不具合が生じるのではと思われますが，その点はマクロが優先で調べられるので，アセンブラは混乱することはありません．

マクロの展開を強制的に終了する命令として，EXITM命令があります．これは，条件疑似命令と組み合わせて使用することにより，アセンブル時間を短縮できます．IFのネストが深くなったとき有効です．

```
「サンプル」
        ANI       MACRO    X,Z
                  IFIDN    <X>,<A>
                  DB       53H,Z
                  ELSE
                  CHECK4   X
                  IF       @FLG
                  DB       98H OR X,Z
                  ELSE
                  ERROR    <INVARID OPERAND>
                -------
                : EXITM :
                -------
                  ENDIF
                  ENDIF
                  ENDM
```

● 反復疑似命令

反復疑似命令は指定した回数だけ繰り返し展開する，マクロに似た疑似命令です．REPT，IRP，IRPCステートメントで始まり，ENDMまたはEXITMで終了します．

```
「サンプル」
        CHECK    MACRO    X
                 @FLG     DEFL     0
                ---------------------------------
               | IRP      Y,<A,R0,R1,R2,R3,     |
               |      R4,R5,R6,R7,@R0,@R1>      |
               | IFIDN    <X>,<Y>               |
               | @FLG     DEFL     1            |
               | EXITM                          |
               | ENDIF                          |
               | ENDM                           |
                ---------------------------------
                 ENDM
```

このCHECKマクロは，パラメータ“X”が8048レジスタ名の範囲にあるかチェックして，範囲にあれば@FLGを“1”セットし，なければ@FLGを“0”にセットしてマクロから抜けます．

● 特殊なマクロ演算子

このマクロ・ライブラリで使用している，マクロ定義の中で使用できる特殊な演算子について説明します．

;;―展開を行わないコメント

```
「サンプル」
        ;; 8048 MACRO LIBRARY
```

LALL命令の実行後でも，(;;)以降のコメントは，マクロ展開のリストに出力されません．

%―マクロのパラメータとして指定した〈式〉を計算し，その値に対するASCII文字をパラメータとします．

```
「サンプル」
        PRINTE   MACRO
                 IF2
                 IF       ERR
                ---------------------------------------
               | PRINTE1  %ERR,< ERROR(s) >          |
                ---------------------------------------
                 ELSE
                 PRINTE1  < NO >,< ERROR(s) >
                 ENDIF
                 ENDIF
                 ENDM
        PRINTE1  MACRO    X,Y
                 .PRINTX  *X,Y*
                 ENDM
```

&―テキストまたはシンボルを連結します．

```
「サンプル」
        @PUT     MACRO    X
                ---------------------------
               | @RE&X     DEFL      $   |
                ---------------------------
                 @CT       DEFL      @CT+1
                 ENDM
        REPEAT   MACRO
                 @PUT      %@CT
                 ENDM
```

このサンプルはプログラム制御，REPEAT－UNTILマクロの一部で，シンボル“@RE”とダミー・パラメータのネスティング・カウンタ“@CT”を連結し，REPEATの置かれたカレント・プログラム・アドレスの値を与えます．次に，プログラム例とアセンブル結果を示します．

```
「プログラム例」
          MVI      R0,10
          REPEAT
                   MVI      R1,100
                   REPEAT
                            DEC      R1
                   UNTIL    R1,EQ,0
                   DEC      R0
          UNTIL    R0,EQ,0
「アセンブル結果」
                        MVI      R0,10
0200   B8 0A    +       DB       0B0H OR R0,10
                        REPEAT
                                 MVI      R1,100
0202   B9 64    +       DB       0B0H OR R1,100
                                 REPEAT
                                          DEC      R1
0204   C9       +       DB       0C0H OR R1
                                 UNTIL    R1,EQ,0
0205   23 00    +       DB       23H,0
0207   37 17    +       DB       37H,17H
0209   69       +       DB       60H OR R1
020A   96 04    +       DB       96H,(@WK AND 0FFH)
                                 DEC      R0
020C   C8       +       DB       0C0H OR R0
                        UNTIL    R0,EQ,R0
020D   23 00    +       DB       23H,0
020F   37 17    +       DB       37H,17H
0211   68       +       DB       60H OR R0
0212   96 02    +       DB       96H,(@WK AND 0FFH)

Symbols:
0202     @RE0      0204     @RE1
```

以上でマクロ・アセンブラの解説を終わりますが，これはMACRO80の機能の一部にすぎません．その他多くの機能については，MACRO80ユーティリティ・ソフトウェア・マニュアルを参照してください．

8048クロス・アセンブラの作り方

8048などの1チップ・マイコンは比較的簡単な命令体系で，プログラム容量が1K～4Kバイト程度です．そこでMACRO80のマクロ機能を使用して，1チップ・マイコンのマシン・コードを発生する，クロス・アセンブラを作ることができます．

問題点として，アセンブル時間が通常のクロス・アセンブラ(XASM48など)よりかかります．これはアセンブラ内部での演算およびマクロ展開に時間を要するからで，ある程度やむをえないことです．

マクロ機能を使用したクロス・アセンブラを，アセンブラ内部のマシン・コード発生機能は使わず，DB(Define Byte)命令とアセンブラの演算機能を利用して，8048が理解できるコードを発生します．

8048の命令をマクロ定義する前に，各命令(8048のマクロ)から参照するサポート・マクロを定義します．サポート・マクロとしては，与えられたパラメータの

レジスタ名およびポート・ナンバが8048の範囲にあるかをチェックするマクロ，分岐命令やCALL命令から参照されるアドレスの演算および範囲をチェックするマクロ，マクロ内部で発生したエラー処理用のマクロなどがあります．

ここで，2～3の命令を例にあげてマクロを作りながら解説します．8048の命令をマクロ定義するうえで三つに分類すれば，

(1) オペランドをもたない命令
　（ＲＥＴ,ＲＥＴＲ,ＮＯＰなど）
(2) オペランドの範囲がかぎられる命令
　（ローテイト命令やフラグ操作命令など）
(3) オペランドをもった命令
　（ＩＮＣ,ＭＯＶなど，大部分の命令）

(1)および(2)の場合は簡単で，マニュアルのマシン・コード表を見ながらマクロ定義すれば作れます．

```
「サンプル」
        RETR    MACRO
                DB      93H
                ENDM
        RL      MACRO   X
                IFIDN   <X>,<A>
                DB      0E7H
                ELSE
                ERROR   <INVALID OPERAND>
                ENDIF
                ENDM
```

(3)の場合は，マクロのネストも深くなり複雑です．INC命令のマシン・コードを見ると，下記のようになっています．INC A 命令は１７Ｈの固定コードをもち，INC Rr 命令は下位３ビットでレジスタ名を表し，間接アドレス指定の INC @Rr 命令は下位ビットで，＠ＲＯ,＠Ｒ１を表しています．マクロの作り方にも，いろいろありますが，一例を示せば，

「ＩＮＣのマシン・コード」

```
            -------------------
INC A      : 0 0 0 1 : 0 1 1 1 :
            -------------------

            -------------------
INC Rr     : 0 0 0 1 : 0 r r r :
            -------------------

            -------------------
INC @Rr    : 0 0 0 1 : 0 0 0 r :
            -------------------
```

```
「サンプル」
        INC     MACRO   X
                IFIDN   <X>,<A>
                DB      17H
                ELSE
                IFIDN   <X>,<R0>
                DB      18H
                ELSE
                IFIDN   <X>,<R1>
                DB      19H
                ELSE
                IFIDN   <X>,<R2>
                DB      1AH
                  :
                  :
                ELSE
                IFIDN   <X>,<@R0>
                DB      10H
                ELSE
                IFIDN   <X>,<@R1>
                DB      11H
                ELSE
                ERROR   <INVALID OPERAND>
                EXITM
                ENDIF
                ENDIF
                  :
                  :
                ENDIF
                ENDIF
                ENDIF
                ENDIF
                ENDM
```

この例では，IFのネストが深くマクロが大きくなります．従ってリストが見づらく，アセンブル時間もかかります．そこで，あらかじめEQU命令でレジスタ名に値を与えておき，演算でマシン・コードを得ます．INC Rr 命令でわかるように，下位３ビットでレジスタ名を表していますが，ここではほかの命令と加味して４ビットで区切り EQU定義しています．

```
「サンプル」
        A       EQU     07H
        R0      EQU     08H
        R1      EQU     09H
        R2      EQU     0AH
        R3      EQU     0BH
        R4      EQU     0CH
        R5      EQU     0DH
        R6      EQU     0EH
        R7      EQU     0FH
        @R0     EQU     00H
        @R1     EQU     01H

        INC     MACRO   X
                CHECKO  X
        IF      @FLG
        DB      10H OR X
        ELSE
        ERROR   <INVALID OPERAND>
        EXITM
        ENDIF
        ENDM
```

上記のように，マクロが短くなり，見やすくなります．ＣＨＥＣＫＯはレジスタ名をチェックするマクロで，ＥＲＲＯＲはエラー処理用のマクロです．マクロ定義の中でエラーが発生したとき，コンソールにエラー・メッセージを出力し，かつリスト出力を抑止されていてもエラーの行をリストに出力します．

またエラー・カウンタをインクリメントしたりする機能があると便利ですが，MACRO80にはありません．そこで，ＥＲＲＯＲマクロを定義し，エラーの発生した箇所のアドレスとエラーの内容をコンソールに出力するようにしています．またソース・リストの最後にＰＲＩＮＴＥを挿入すると，マクロ内部で発生してエラーの数を知ることができます．

ニモニックの相違点について，このマクロ・アセン

ブラは，インテル社の発表しているニモニックのうちイミディエイト命令のみ違います．8048の場合は定数の直前に“#”を付けてシンボルと区別していますが，マクロ定義を簡単にするためにイミディエイト命令を8080Aに似たマクロ名で定義しています．下記に例を示しますが，その他の命令はSAMPLEプログラムを参照してください．

```
「サンプル」
    ADD        A,#12   --->  ADI      A,12
    MOV        A,#12   --->  MVI      A,12
```

マクロ・ライブラリのチェックは，SAMPLEプログラムをアセンブルし，出力されたリスト・ファイルとマニュアルを比較して行います．このマクロ・ライブラリを実際に仕事で何度か使用しましたが，問題なく動いています．

このクロス・アセンブラのリスト・ファイルは，リスト10-1のように各命令の間にマシン・コードを出力するDB命令が入り，大変見づらいリストになります．だからといって(．SALL)でマクロ展開を抑止すると，マシン・コードがリストから消えソース・リストと変らなくなります．そこで，リスト・ファイルを見やすくする，ファイル変換プログラムを作りました．筆者の場合，中野正次氏の「実戦マクロ・アセンブラ活用法」を参考に一部機能を追加しました．このプログラムは頻繁にディスクをアクセスするためバッファを設け，ディスク・アクセスの回数を減らし，また変換に時間がかかるので，進行状況を一目でわかるようにしています．これはTURBO PASCALで作り，使用しています．

8048クロス アセンブラの実行

8048マクロ・ライブラリを使用したプログラムの開発手順を図10-4に示します．始めにエディタを使用して，ソース・プログラムを作成しますが，ソース・プログラムの最初に，SAMPLEプログラムのように，ＩＮＣＬＵＤＥ ８０４８．ＬＩＢ，ＡＳＥＧ，ＯＲＧ ×××Ｈを挿入してください．その他の機能はMACRO80に依存します．クロス・アセンブラを実行するには，下記のファイルが必要です．

Ｍ８０．ＣＯＭ←MACRO80マクロ・アセンブラ

Ｌ８０．ＣＯＭ←LINK80リンク・ローダ

８０４８．ＬＩＢ←8048マクロ・ライブラリ

ＡＳＭ４８．ＳＵＢ←インテルHEXファイル作成までを自動的に処理する．サブミット・ファイル

Ｓ．ＣＯＭ←入力文字数を減らすためリネームしたＳＵＢＭＩＴ．ＣＯＭ

ＸＳＵＢ．ＣＯＭ

ＡＳＭ４８．ＳＵＢの内容を下記に示します．

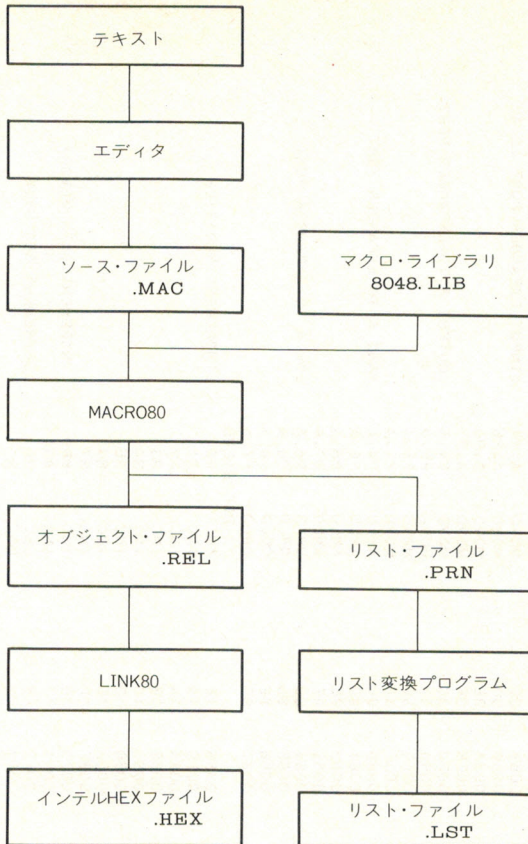
ＳＵＢＭＩＴ．ＣＯＭ，ＸＳＵＢ．ＣＯＭの機能およびサブミット・ファイルの作り方は，CP/Mマニュアルを参照してください．

```
XSUB
M80 =$1$2
L80 $1,$1/X/N/E
N
ERA $1.REL
```

クロス・アセンブラの呼び出し方は，次のとおりで

〈リスト10-1〉 クロス・アセンブラのリスト

```
MACRO－80のリスト・ファイル
0000'                          ASEG
                               ORG       0

                               ADD       A,R0
0000       68           +      DB        60H OR R0
                               ADD       A,R1
0001       69           +      DB        60H OR R1
                               ADD       A,R2
0002       6A           +      DB        60H OR R2
                               ADD       A,R3
0003       6B           +      DB        60H OR R3
                               ADD       A,R4
0004       6C           +      DB        60H OR R4
                               ADD       A,R5
0005       6D           +      DB        60H OR R5
                               ADD       A,R6
0006       6E           +      DB        60H OR R6
                               ADD       A,R7
0007       6F           +      DB        60H OR R7
                               ADD       A,@R0
0008       60           +      DB        60H OR @R0
                               ADD       A,@R1
0009       61           +      DB        60H OR @R1


変換プログラムを実行後のリスト・ファイル
0000'                          ASEG
                               ORG       0

0000       68                  ADD       A,R0
0001       69                  ADD       A,R1
0002       6A                  ADD       A,R2
0003       6B                  ADD       A,R3
0004       6C                  ADD       A,R4
0005       6D                  ADD       A,R5
0006       6E                  ADD       A,R6
0007       6F                  ADD       A,R7
0008       60                  ADD       A,@R0
0009       61                  ADD       A,@R1
```

〈リスト10-4〉 8048マクロ・ライブラリ

す.

```
A>S ASM48 ソース・ファイル [スイッチ]
```

リスト10-2はアセンブル時にコンソールに出力された内容です.

● まとめ

以上，MACRO80のマクロ機能を使用した8048クロス・アセンブラを紹介しました. 1チップ・マイコンは今後さらに低価格になり，各機器に応用されると思われます. 68系や8048シリーズとアップ・コンパチブルの8051などのクロス・アセンブラも比較的簡単に作れ，十分実用になります. 1チップ・マイコンの導入を考えている方やMACRO80の本格的な応用を考えている方の参考になれば幸いです.

◈参考・引用*文献◈

(1) ユーティリティ・ソフトウェア・マニュアル，アスキー.
(2) インテルジャパン㈱，マイクロコンピュータ，ユーザーズ・マニュアル MCS-48，CQ出版社.
(3) ボーランド・インターナショナル：マイクロソフトウェア，TURBO PASCAL プログラミング・マニュアル.
(4)* 中野正次；実戦マクロ・アセンブラ活用法，CQ出版社.
(5) 前田英明；マクロ・アセンブラの使い方，工学図書.
(6)*Z80ファミリ テクニカルマニュアル，1980年，シャープ㈱.
(7) M80ユーティリティ・ソフトウェア手引書，Microsoft，1978年.
(8) 最新マイコン周辺LSI規格表，CQ出版社，1987年.
(9) 酒井重恭；コンピュータ用語の基礎知識，共立出版.

〈リスト10-2〉 アセンブルの様子

```
*A>S ASM48 SAMPLE /L

*A>XSUB

*A>M80 =SAMPLE/L
* NO , ERROR(s) *  <--  マクロ内部のエラー数

No Fatal error(s)  <--  MACRO-80のエラー数

*A>L80 SAMPLE,SAMPLE/X/N/E

Linh-80  3.44  09-Dec-81  Copyright (c) 1981 Microsoft
%Overlaying Program area

Data    0000    0115    <  277>

50469 Byter Free
[0000   0115         1]
Origin below loader memory, move anyway(Y or N)?N

*A>ERA SAMPLE.REL

*A>
```

〈リスト10-3〉サンプル・プログラム

```
                        ;
                        ; 8048 INSTRUCTION SET
                        ;
                        TITLE SAMPLE
                        IF1
                        INCLUDE 8048.LIB
                        ENDIF
0123                    ADDRESS EQU     123H
0045                    DATA    EQU     45H
0000'                   ASEG
                        ORG     0H
                        ;  MNEMONIC
                        ;( ACCUMULATOR )
0000    68              ADD     A,R0    ;ADD REGISTER TO A
0001    69              ADD     A,R1
0002    6A              ADD     A,R2
0003    6B              ADD     A,R3
0004    6C              ADD     A,R4
0005    6D              ADD     A,R5
0006    6E              ADD     A,R6
0007    6F              ADD     A,R7
0008    60              ADD     A,@R0   ;ADD DATA MEMORY TO A
0009    61              ADD     A,@R1
000A    03 45           ADI     A,DATA  ;ADD IMMEDIATE TO A
000C    78              ADDC    A,R0    ;ADD REGISTER WITH CARRY
000D    79              ADDC    A,R1
000E    7A              ADDC    A,R2
000F    7B              ADDC    A,R3
0010    7C              ADDC    A,R4
0011    7D              ADDC    A,R5
0012    7E              ADDC    A,R6
0013    7F              ADDC    A,R7
0014    70              ADDC    A,@R0   ;ADD DATA MEMORY WITH CARRY
0015    71              ADDC    A,@R1
0016    13 45           ACI     A,DATA  ;ADD IMMEDIATE WITH CARRY
0018    58              ANL     A,R0    ;AND REGISTER TO A
0019    59              ANL     A,R1
001A    5A              ANL     A,R2
001B    5B              ANL     A,R3
001C    5C              ANL     A,R4
001D    5D              ANL     A,R5
001E    5E              ANL     A,R6
001F    5F              ANL     A,R7
0020    50              ANL     A,@R0   ;AND DATA MEMORY TO A
0021    51              ANL     A,@R1
0022    53 45           ANI     A,DATA  ;AND IMMEDIATE TO A
0024    48              ORL     A,R0    ;OR REGISTER TO A
0025    49              ORL     A,R1
0026    4A              ORL     A,R2
0027    4B              ORL     A,R3
0028    4C              ORL     A,R4
0029    4D              ORL     A,R5
002A    4E              ORL     A,R6
002B    4F              ORL     A,R7
002C    40              ORL     A,@R0   ;OR DATA MEMORY TO A
002D    41              ORL     A,@R1
002E    43 45           ORI     A,DATA  ;OR IMMEDIATE TO A
0030    D8              XRL     A,R0    ;EXCLUSIVE OR REGISTER TO A
0031    D9              XRL     A,R1
0032    DA              XRL     A,R2
0033    DB              XRL     A,R3
0034    DC              XRL     A,R4
0035    DD              XRL     A,R5
0036    DE              XRL     A,R6
0037    DF              XRL     A,R7
0038    D0              XRL     A,@R0   ;EXCLUSIVE OR DATA MEMORY TO A
0039    D1              XRL     A,@R1
003A    D3 45           XRI     A,DATA  ;EXCLUSIVE OR IMMEDIATE TO A
003C    17              INC     A       ;INCREMENT A
003D    07              DEC     A       ;DECREMENT A
003E    27              CLR     A       ;CLEAR A
003F    37              CPL     A       ;COMPLEMENT A
0040    57              DA      A       ;DECIMAL ADJUST A
0041    47              SWAP    A       ;SWAP NIBBLES OF A
0042    E7              RL      A       ;ROTATE A LEFT
0043    F7              RLC     A       ;ROTATE A LEFT THROUGH CARRY
0044    77              RR      A       ;ROTATE A RIGHT
0045    67              RRC     A       ;ROTATE A RIGHT THROUGH CARRY
                        ;( INPUT / OUTPUT )
0046    09              IN      A,P1    ;INPUT PORT TO A
0047    0A              IN      A,P2
0048    39              OUTL    P1,A    ;OUTPUT A TO PORT
0049    3A              OUTL    P2,A
004A    99 45           ANI     P1,DATA ;AND IMMEDIATE TO PORT
004C    9A 45           ANI     P2,DATA
004E    89 45           ORI     P1,DATA ;OR IMMEDIATE TO PORT
0050    8A 45           ORI     P2,DATA
0052    08              INS     A,BUS   ;INPUT BUS TO A
0053    02              OUTL    BUS,A   ;OUTPUT A TO BUS
0054    98 45           ANI     BUS,DATA;AND IMMEDIATE TO BUS
0056    88 45           ORI     BUS,DATA;OR IMMEDIATE TO BUS
0058    0C              MOVD    A,P4    ;INPUT EXPANDER PORT TO A
0059    0D              MOVD    A,P5
005A    0E              MOVD    A,P6
005B    0F              MOVD    A,P7
005C    3C              MOVD    P4,A    ;OUTPUT A TO EXPANDER PORT
005D    3D              MOVD    P5,A
005E    3E              MOVD    P6,A
005F    3F              MOVD    P7,A
0060    9C              ANLD    P4,A    ;AND A TO EXPANDER PORT
0061    9D              ANLD    P5,A
0062    9E              ANLD    P6,A
0063    9F              ANLD    P7,A
0064    8C              ORLD    P4,A    ;OR A TO EXPANDER PORT
0065    8D              ORLD    P5,A
0066    8E              ORLD    P6,A
0067    8F              ORLD    P7,A
                        ;( REGISTERS )
0068    18              INC     R0      ;INCREMENT REGISTER
0069    19              INC     R1
006A    1A              INC     R2
006B    1B              INC     R3
006C    1C              INC     R4
006D    1D              INC     R5
006E    1E              INC     R6
006F    1F              INC     R7
0070    10              INC     @R0     ;INCREMENT DATA MEMORY
0071    11              INC     @R1
0072    C8              DEC     R0      ;DECREMENT REGISTER
0073    C9              DEC     R1
```

〈リスト10-3〉サンプル・プログラム(つづき)

```
0074    CA              DEC     R2
0075    CB              DEC     R3
0076    CC              DEC     R4
0077    CD              DEC     R5
0078    CE              DEC     R6
0079    CF              DEC     R7
                        ;( BRANCH )
007A            ADRS:
007A    04 7A           JMP     ADRS    ;JUMP UNCONDITIONAL
007C    B3              JMPP    @A      ;JUMP INDIRECT
007D    E8 7A           DJNZ    R0,ADRS ;DECREMENT REGISTER AND SKIP
007F    E9 7A           DJNZ    R1,ADRS
0081    EA 7A           DJNZ    R2,ADRS
0083    EB 7A           DJNZ    R3,ADRS
0085    EC 7A           DJNZ    R4,ADRS
0087    ED 7A           DJNZ    R5,ADRS
0089    EE 7A           DJNZ    R6,ADRS
008B    EF 7A           DJNZ    R7,ADRS
008D    F6 7A           JC      ADRS    ;JUMP ON CARRY = 1
008F    E6 7A           JNC     ADRS    ;JUMP ON CARRY = 0
0091    C6 7A           JZ      ADRS    ;JUMP ON A ZERO
0093    96 7A           JNZ     ADRS    ;JUMP ON A NOT ZERO
0095    36 7A           JT0     ADRS    ;JUMP ON T0 = 1
0097    26 7A           JNT0    ADRS    ;JUMP ON T0 = 0
0099    56 7A           JT1     ADRS    ;JUMP ON T1 = 1
009B    46 7A           JNT1    ADRS    ;JUMP ON T1 = 0
009D    B6 7A           JF0     ADRS    ;JUMP ON F0 = 1
009F    76 7A           JF1     ADRS    ;JUMP ON F1 = 1
00A1    16 7A           JTF     ADRS    ;JUMP ON TIMER FLAG
00A3    86 7A           JNI     ADRS    ;JUMP ON INT = 0
00A5    12 7A           JB0     ADRS    ;JUMP ON ACCUMULATOR BIT
00A7    32 7A           JB1     ADRS
00A9    52 7A           JB2     ADRS
00AB    72 7A           JB3     ADRS
00AD    92 7A           JB4     ADRS
00AF    B2 7A           JB5     ADRS
00B1    D2 7A           JB6     ADRS
00B3    F2 7A           JB7     ADRS
                        ;( SUBROUTINE )
00B5    34 23           CALL    ADDRESS ;JUMP TO SUBROUTINE
00B7    83              RET             ;RETURN
00B8    93              RETR            ;RETURN AND RESTORE STATUS
                        ;( FLAGS )
00B9    97              CLR     C       ;CLEAR CARRY
00BA    A7              CPL     C       ;COMPLEMENT CARRY
00BB    85              CLR     F0      ;CLEAR FLAG 0
00BC    95              CPL     F0      ;COMPLEMENT FLAG 0
00BD    A5              CLR     F1      ;CLEAR FLAG 1
00BE    B5              CPL     F1      ;COMPLEMENT FLAG 1
                        ;( DATA MOVES )
00BF    F8              MOV     A,R0    ;MOVE REGISTER TO A
00C0    F9              MOV     A,R1
00C1    FA              MOV     A,R2
00C2    FB              MOV     A,R3
00C3    FC              MOV     A,R4
00C4    FD              MOV     A,R5
00C5    FE              MOV     A,R6
00C6    FF              MOV     A,R7
00C7    F0              MOV     A,@R0   ;MOVE DATA MEMORY TO A
00C8    F1              MOV     A,@R1
00C9    23 45           MVI     A,DATA  ;MOVE IMMEDIATE TO A
00CB    A8              MOV     R0,A    ;MOVE A TO TOREGISTER
00CC    A9              MOV     R1,A
00CD    AA              MOV     R2,A
00CE    AB              MOV     R3,A
00CF    AC              MOV     R4,A
00D0    AD              MOV     R5,A
00D1    AE              MOV     R6,A
00D2    AF              MOV     R7,A
00D3    A0              MOV     @R0,A   ;MOVE A TO DATA MEMORY
00D4    A1              MOV     @R1,A
00D5    B8 45           MVI     R0,DATA ;MOVE IMMEDIATE TO REGISTER
00D7    B9 45           MVI     R1,DATA
00D9    BA 45           MVI     R2,DATA
00DB    BB 45           MVI     R3,DATA
00DD    BC 45           MVI     R4,DATA
00DF    BD 45           MVI     R5,DATA
00E1    BE 45           MVI     R6,DATA
00E3    BF 45           MVI     R7,DATA
00E5    B0 45           MVI     @R0,DATA;MOVE IMMEDIATE TO DATA MEMORY
00E7    B1 45           MVI     @R1,DATA
00E9    C7              MOV     A,PSW   ;MOVE PSW TO A
00EA    D7              MOV     PSW,A   ;MOVE A TO PSW
00EB    28              XCH     A,R0    ;EXCHANGE A AND REGISTER
00EC    29              XCH     A,R1
00ED    2A              XCH     A,R2
00EE    2B              XCH     A,R3
00EF    2C              XCH     A,R4
00F0    2D              XCH     A,R5
00F1    2E              XCH     A,R6
00F2    2F              XCH     A,R7
00F3    20              XCH     A,@R0   ;EXCHANGE A AND DATA MEMORY
00F4    21              XCH     A,@R1
00F5    30              XCHD    A,@R0   ;EXCHANGE NIBBLE OF A AND DATA MEMORY
00F6    31              XCHD    A,@R1
00F7    80              MOVX    A,@R0   ;MOVE EXTERNAL DATA MEMORY TO A
00F8    81              MOVX    A,@R1
00F9    90              MOVX    @R0,A   ;MOVE A TO EXTERNAL DATA MEMORY
00FA    91              MOVX    @R1,A
00FB    A3              MOVP    A,@A    ;MOVE TO A FROM CURRNT PAGE
00FC    E3              MOVP3   A,@A    ;MOVE TO A FROM PAGE 3
                        ;( TIMER / COUNTER )
00FD    42              MOV     A,T     ;READ TIMER/COUNTER
00FE    62              MOV     T,A     ;LOAD TIMER/COUNTER
00FF    55              STRT    T       ;START TINER
0100    45              STRT    CNT     ;START COUNTER
0101    65              STOP    TCNT    ;STOP TIMER/COUNTER
0102    25              EN      TCNTI   ;ENABLE TIMER/COUNIER INTERRUPT
0103    35              DIS     TCNTI   ;DISABLE TIMER/COUNTER INTERRUPT
                        ;( CONTROL )
0104    05              EN      I       ;ENABLE EXTERNAL INTERRUPT
0105    15              DIS     I       ;DISABLE EXTERNAL INTERRUPT
0106    C5              SEL     RB0     ;SELECT REGISTER BANK 0
0107    D5              SEL     RB1     ;SELECT REGISTER BANK 1
0108    E5              SEL     MB0     ;SELECT MEMORY BANK 0
0109    F5              SEL     MB1     ;SELECT MEMORY BANK 1
010A    75              ENT0    CLK     ;ENABLE CLOCK OUTPUT ON T0
                        ;
010B    00              NOP             ;NO OPERATION

                        PRINTE
                        END
```

〈リスト10-4〉 8048マクロ・ライブラ

```
 4:
 5:         ;;
 6:         ;; 8048 MACRO LIBRARY
 7:         ;;
 8:         ;; '85/09/15  S.KASHIRO
 9:         ;;
10: CHECK0  MACRO   X
11:         @FLG    DEFL    0
12:         IRP     Y,<A,R0,R1,R2,R3,R4,R5,R6,R7,@R0,@R1>
13:         IFIDN   <X>,<Y>
14:         @FLG    DEFL    1
15:         EXITM
16:         ENDIF
17:         ENDM
18:         ENDM
19: CHECK1  MACRO   X
20:         @FLG    DEFL    0
21:         IRP     Y,<R0,R1,R2,R3,R4,R5,R6,R7,@R0,@R1>
22:         IFIDN   <X>,<Y>
23:         @FLG    DEFL    1
24:         EXITM
25:         ENDIF
26:         ENDM
27:         ENDM
28: CHECK2  MACRO   X
29:         @FLG    DEFL    0
30:         IRP     Y,<R0,R1,R2,R3,R4,R5,R6,R7>
31:         IFIDN   <X>,<Y>
32:         @FLG    DEFL    1
33:         EXITM
34:         ENDIF
35:         ENDM
36:         ENDM
37: CHECK3  MACRO   X
38:         @FLG    DEFL    0
39:         IRP     Y,<@R0,@R1>
40:         IFIDN   <X>,<Y>
41:         @FLG    DEFL    1
42:         EXITM
43:         ENDIF
44:         ENDM
45:         ENDM
46: CHECK4  MACRO   X
47:         @FLG    DEFL    0
48:         IRP     Y,<BUS,P1,P2>
49:         IFIDN   <X>,<Y>
50:         @FLG    DEFL    1
51:         EXITM
52:         ENDIF
53:         ENDM
54:         ENDM
55: CHECK5  MACRO   X
56:         @FLG    DEFL    0
57:         IRP     Y,<P4,P5,P6,P7>
58:         IFIDN   <X>,<Y>
59:         @FLG    DEFL    1
60:         EXITM
61:         ENDIF
62:         ENDM
63:         ENDM
64: CHECKA  MACRO   X,Y
```

```
 65:         IFIDN   <X>,<A>
 66:         DB      Y
 67:         ELSE
 68:         ERROR   <INVALID OPERAND>
 69:         EXITM
 70:         ENDIF
 71:         ENDM
 72: SHORT   MACRO   X,Y
 73:         DB      X,(Y AND 0FFH)
 74:         IF2
 75:         IF      (($ SHR 8)-(Y SHR 8))
 76:         ERROR   <OPERAND FEILD.>
 77:         ENDIF
 78:         ENDIF
 79:         ENDM
 80: LONG    MACRO   X,Y
 81:         DB      X OR ((Y SHR 3) AND 0E0H),Y AND 0FFH
 82:         ENDM
 83: @PUT    MACRO   X
 84:         @RE&X   DEFL    $
 85:         @CT     DEFL    @CT+1
 86:         ENDM
 87: @GET    MACRO   X
 88:         @WK     DEFL    @RE&X
 89:         ENDM
 90: ERROR   MACRO   Y
 91:         IF2
 92:         .RADIX  16
 93: @LOC    DEFL    $
 94:         PRINT$  %@LOC,<Y>
 95: @ERR    DEFL    @ERR+1
 96:         .RADIX  10
 97:         ENDIF
 98:         ENDM
 99: PRINT$  MACRO   X,Y
100:         .PRINTX *X,Y*
101:         ENDM
102: RET     MACRO
103:         DB      83H
104:         ENDM
105: RETR    MACRO
106:         DB      93H
107:         ENDM
108: INC     MACRO   X
109:         CHECK0  X
110:         IF      @FLG
111:         DB      10H OR X
112:         ELSE
113:         ERROR   <INVALID OPERAND>
114:         EXITM
115:         ENDIF
116:         ENDM
117: DEC     MACRO   X
118:         IFIDN   <X>,<A>
119:         DB      7H
120:         ELSE
121:         CHECK2  X
122:         IF      @FLG
123:         DB      0C0H OR X
124:         ELSE
125:         ERROR   <INVALID OPERAND>
```

```
126:         EXITM
127:         ENDIF
128:         ENDIF
129:         ENDM
130: ADD     MACRO   X,Y
131:         IFIDN   <X>,<A>
132:         CHECK1  Y
133:         IF      @FLG
134:         DB      60H OR Y
135:         ELSE
136:         ERROR   <INVALID OPERAND>
137:         EXITM
138:         ENDIF
139:         ELSE
140:         ERROR   <INVALID OPERAND>
141:         ENDIF
142:         ENDM
143: ADDC    MACRO   X,Y
144:         IFIDN   <X>,<A>
145:         CHECK1  Y
146:         IF      @FLG
147:         DB      70H OR Y
148:         ELSE
149:         ERROR   <INVALID OPERAND>
150:         EXITM
151:         ENDIF
152:         ELSE
153:         ERROR   <INVALID OPERAND>
154:         ENDIF
155:         ENDM
156: ANL     MACRO   X,Y
157:         IFIDN   <X>,<A>
158:         CHECK1  Y
159:         IF      @FLG
160:         DB      50H OR Y
161:         ELSE
162:         ERROR   <INVALID OPERAND>
163:         EXITM
164:         ENDIF
165:         ELSE
166:         ERROR   <INVALID OPERAND>
167:         ENDIF
168:         ENDM
169: ORL     MACRO   X,Y
170:         IFIDN   <X>,<A>
171:         CHECK1  Y
172:         IF      @FLG
173:         DB      40H OR Y
174:         ELSE
175:         ERROR   <INVALID OPERAND>
176:         EXITM
177:         ENDIF
178:         ELSE
179:         ERROR   <INVALID OPERAND>
180:         ENDIF
181:         ENDM
182: XRL     MACRO   X,Y
183:         IFIDN   <X>,<A>
184:         CHECK1  Y
185:         IF      @FLG
186:         DB      0D0H OR Y
```

〈リスト10-4〉 8048マクロ・ライブラ(つづき)

```
187:         ELSE
188:         ERROR   <INVALID OPERAND>
189:         EXITM
190:         ENDIF
191:         ELSE
192:         ERROR   <INVALID OPERAND>
193:         ENDIF
194:         ENDM
195:  XCH    MACRO   X,Y
196:         IFIDN   <X>,<A>
197:         CHECK1  Y
198:         IF      @FLG
199:         DB      20H OR Y
200:         ELSE
201:         ERROR   <INVALID OPERAND>
202:         EXITM
203:         ENDIF
204:         ELSE
205:         ERROR   <INVALID OPERAND>
206:         ENDIF
207:         ENDM
208:  XCHD   MACRO   X,Y
209:         IFIDN   <X>,<A>
210:         CHECK3  Y
211:         IF      @FLG
212:         DB      30H OR Y
213:         ELSE
214:         ERROR   <INVALID OPERAND>
215:         EXITM
216:         ENDIF
217:         ELSE
218:         ERROR   <INVALID OPERAND>
219:         ENDIF
220:         ENDM
221:  ADI    MACRO   X,Z
222:         IFIDN   <X>,<A>
223:         DB      03H,Z
224:         ELSE
225:         ERROR   <INVALID OPERAND>
226:         ENDIF
227:         ENDM
228:  ACI    MACRO   X,Z
229:         IFIDN   <X>,<A>
230:         DB      13H,Z
231:         ELSE
232:         ERROR   <INVALID OPERAND>
233:         ENDIF
234:         ENDM
235:  ANI    MACRO   X,Z
236:         IFIDN   <X>,<A>
237:         DB      53H,Z
238:         ELSE
239:         CHECK4  X
240:         IF      @FLG
241:         DB      98H OR X,Z
242:         ELSE
243:         ERROR   <INVALID OPERAND>
244:         ENDIF
245:         ENDIF
246:         ENDM
247:  ORI    MACRO   X,Z
248:         IFIDN   <X>,<A>
249:         DB      43H,Z
250:         ELSE
251:         CHECK4  X
252:         IF      @FLG
253:         DB      88H OR X,Z
254:         ELSE
255:         ERROR   <INVALID OPERAND>
256:         ENDIF
257:         ENDIF
258:         ENDM
259:  XRI    MACRO   X,Z
260:         IFIDN   <X>,<A>
261:         DB      0D3H,Z
262:         ELSE
263:         ERROR   <INVALID OPERAND>
264:         ENDIF
265:         ENDM
266:  MOVX   MACRO   X,Y
267:         IFIDN   <X>,<A>
268:         CHECK3  Y
269:         IF      @FLG
270:         DB      80H OR Y
271:         ELSE
272:         ERROR   <INVALID OPERAND>
273:         ENDIF
274:         ELSE
275:         IFIDN   <Y>,<A>
276:         CHECK3  X
277:         IF      @FLG
278:         DB      90H OR X
279:         ELSE
280:         ERROR   <INVALID OPERAND>
281:         ENDIF
282:         ENDIF
283:         ENDIF
284:         ENDM
285:  MOVP3  MACRO   X,Y
286:         IFIDN   <X>,<A>
287:         IFIDN   <Y>,<@A>
288:         DB      0E3H
289:         ENDIF
290:         ELSE
291:         ERROR   <INVALID OPERAND>
292:         EXITM
293:         ENDIF
294:         ENDM
295:  MOVP   MACRO   X,Y
296:         IFIDN   <X>,<A>
297:         IFIDN   <Y>,<@A>
298:         DB      0A3H
299:         ENDIF
300:         ELSE
301:         ERROR   <INVALID OPERAND>
302:         EXITM
303:         ENDIF
304:         ENDM
305:  MOV    MACRO   X,Y
306:         IFIDN   <X>,<A>
307:         CHECK1  Y
308:         IF      @FLG
309:         DB      0F0H OR Y
310:         ELSE
311:         IFIDN   <Y>,<T>
312:         DB      42H
313:         ELSE
314:         IFIDN   <Y>,<PSW>
315:         DB      0C7H
316:         ELSE
317:         ERROR   <INVALID OPERAND>
318:         EXITM
319:         ENDIF
320:         ENDIF
321:         ENDIF
322:         ELSE
323:         IFIDN   <Y>,<A>
324:         CHECK1  X
325:         IF      @FLG
326:         DB      0A0H OR X
327:         ELSE
328:         IFIDN   <X>,<PSW>
329:         DB      0D7H
330:         ELSE
331:         IFIDN   <Y>,<A>
332:         DB      62H
333:         ELSE
334:         ERROR   <INVALID OPERAND>
335:         EXITM
336:         ENDIF
337:         ENDIF
338:         ENDIF
339:         ENDIF
340:         ENDIF
341:         ENDM
342:  MVI    MACRO   X,Y
343:         IFIDN   <X>,<A>
344:         DB      23H,Y
345:         ELSE
346:         CHECK1  X
347:         IF      @FLG
348:         DB      0B0H OR X,Y
349:         ELSE
350:         ERROR   <INVALID OPERAND>
351:         ENDIF
352:         ENDIF
353:         ENDM
354:  STOP   MACRO   X
355:         IFIDN   <X>,<TCNT>
356:         DB      65H
357:         ELSE
358:         ERROR   <INVALID OPERAND>
359:         ENDIF
360:         ENDM
361:  STRT   MACRO   X
362:         IFIDN   <X>,<T>
363:         DB      55H
364:         ELSE
365:         IFIDN   <X>,<CNT>
366:         DB      45H
367:         ELSE
368:         ERROR   <INVALID OPERAND>
369:         ENDIF
```

〈リスト10-4〉 8048マクロ・ライブラ(つづき)

```
370:         ENDIF
371:         ENDM
372:  JMP    MACRO   Y
373:         LONG    4,Y
374:         ENDM
375:  CALL   MACRO   Y
376:         LONG    14H,Y
377:         ENDM
378:  JMPP   MACRO   X
379:         IFIDN   <X>,<@A>
380:         DB      0B3H
381:         ELSE
382:         ERROR   <INVALID OPERAND>
383:         EXITM
384:         ENDIF
385:         ENDM
386:  JC     MACRO   Y
387:         SHORT   0F6H,Y
388:         ENDM
389:  JF0    MACRO   Y
390:         SHORT   0B6H,Y
391:         ENDM
392:  JF1    MACRO   Y
393:         SHORT   76H,Y
394:         ENDM
395:  JNC    MACRO   Y
396:         SHORT   0E6H,Y
397:         ENDM
398:  JNI    MACRO   Y
399:         SHORT   86H,Y
400:         ENDM
401:  JNT0   MACRO   Y
402:         SHORT   26H,Y
403:         ENDM
404:  JNT1   MACRO   Y
405:         SHORT   46H,Y
406:         ENDM
407:  JNZ    MACRO   Y
408:         SHORT   96H,Y
409:         ENDM
410:  JTF    MACRO   Y
411:         SHORT   16H,Y
412:         ENDM
413:  JT0    MACRO   Y
414:         SHORT   36H,Y
415:         ENDM
416:  JT1    MACRO   Y
417:         SHORT   56H,Y
418:         ENDM
419:  JZ     MACRO   Y
420:         SHORT   0C6H,Y
421:         ENDM
422:  JB0    MACRO   Y
423:         SHORT   12H,Y
424:         ENDM
425:  JB1    MACRO   Y
426:         SHORT   32H,Y
427:         ENDM
428:  JB2    MACRO   Y
429:         SHORT   52H,Y
430:         ENDM
431:  JB3    MACRO   Y
432:         SHORT   72H,Y
433:         ENDM
434:  JB4    MACRO   Y
435:         SHORT   92H,Y
436:         ENDM
437:  JB5    MACRO   Y
438:         SHORT   0B2H,Y
439:         ENDM
440:  JB6    MACRO   Y
             SHORT   0D2H,Y
441:         ENDM
442:  JB7    MACRO   Y
443:         SHORT   0F2H,Y
444:         ENDM
445:  DJNZ   MACRO   X,Y
446:         CHECK2  X
447:         IF      @FLG
448:         SHORT   <0E0H OR X>,Y
449:         ELSE
450:         ERROR   <INVALID OPERAND>
451:         ENDIF
452:         ENDM
453:  IN     MACRO   X,Y
454:         IFIDN   <X>,<A>
455:         IFIDN   <Y>,<P1>
456:         DB      9H
457:         ELSE
458:         IFIDN   <Y>,<P2>
459:         DB      0AH
460:         ELSE
461:         ERROR   <INVALID OPERAND>
462:         EXITM
463:         ENDIF
464:         ENDIF
465:         ENDIF
466:         ENDM
467:  OUTL   MACRO   X,Y
468:         IFDIF   <Y>,<A>
469:         ERROR   <INVALID OPERAND>
470:         ELSE
471:         IFIDN   <X>,<P1>
472:         DB      39H
473:         ELSE
474:         IFIDN   <X>,<P2>
475:         DB      3AH
476:         ELSE
477:         IFIDN   <X>,<BUS>
478:         DB      02H
479:         ELSE
480:         ERROR   <INVALID OPERAND>
481:         EXITM
482:         ENDIF
483:         ENDIF
484:         ENDIF
485:         ENDIF
486:         ENDM
487:  INS    MACRO   X,Y
488:         IFDIF   <X>,<A>
489:         ERROR   <INVALID OPERAND>
490:         ELSE
491:         IFIDN   <Y>,<BUS>
492:         DB      8H
493:         ELSE
494:         ERROR   <INVALID OPERAND>
495:         EXITM
496:         ENDIF
497:         ENDIF
498:         ENDM
499:  MOVD   MACRO   X,Y
500:         IFIDN   <X>,<A>
501:         CHECK5  Y
502:         IF      @FLG
503:         DB      0CH OR Y
504:         ELSE
505:         ERROR   <INVALID OPERAND>
506:         ENDIF
507:         EXITM
508:         ELSE
509:         IFIDN   <Y>,<A>
510:         CHECK5  X
511:         IF      @FLG
512:         DB      3CH OR X
513:         ELSE
514:         ERROR   <INVALID OPERAND>
515:         ENDIF
516:         ENDIF
517:         ENDIF
518:         ENDM
519:  ANLD   MACRO   X,Y
520:         IFDIF   <Y>,<A>
521:         ERROR   <INVALID OPERAND>
522:         EXITM
523:         ELSE
524:         CHECK5  X
525:         IF      @FLG
526:         DB      9CH OR X
527:         ELSE
528:         ERROR   <INVALID OPERAND>
529:         ENDIF
530:         ENDIF
531:         ENDM
532:  ORLD   MACRO   X,Y
533:         IFDIF   <Y>,<A>
534:         ERROR   <INVALID OPERAND>
535:         EXITM
536:         ELSE
537:         CHECK5  X
538:         IF      @FLG
539:         DB      8CH OR X
540:         ELSE
541:         ERROR   <INVALID OPERAND>
542:         ENDIF
543:         ENDIF
544:         ENDM
545:  DA     MACRO   X
546:         CHECKA  X,57H
547:         ENDM
548:  SWAP   MACRO   X
549:         CHECKA  X,47H
550:         ENDM
551:  RRC    MACRO   X
552:         CHECKA  X,67H
553:         ENDM
554:  RR     MACRO   X
555:         CHECKA  X,77H
556:         ENDM
557:  RLC    MACRO   X
```

〈リスト10-4〉8048マクロ・ライブラ(つづき)

```
558:            CHECKA  X,0F7H
559:            ENDM
560:    RL      MACRO   X
561:            CHECKA  X,0E7H
562:            ENDM
563:    CLR     MACRO   X
564:            IFIDN   <X>,<C>
565:            DB      97H
566:            ELSE
567:            IFIDN   <X>,<F0>
568:            DB      85H
569:            ELSE
570:            IFIDN   <X>,<F1>
571:            DB      0A5H
572:            ELSE
573:            IFIDN   <X>,<A>
574:            DB      27H
575:            ELSE
576:            ERROR   <INVALID OPERAND>
577:            EXITM
578:            ENDIF
579:            ENDIF
580:            ENDIF
581:            ENDIF
582:            ENDM
583:    CPL     MACRO   X
584:            IFIDN   <X>,<C>
585:            DB      0A7H
586:            ELSE
587:            IFIDN   <X>,<F0>
588:            DB      95H
589:            ELSE
590:            IFIDN   <X>,<F1>
591:            DB      0B5H
592:            ELSE
593:            IFIDN   <X>,<A>
594:            DB      37H
595:            ELSE
596:            ERROR   <INVALID OPERAND>
597:            EXITM
598:            ENDIF
599:            ENDIF
600:            ENDIF
601:            ENDIF
602:            ENDM
603:    SEL     MACRO   X
604:            IFIDN   <X>,<RB0>
605:            DB      0C5H
606:            ELSE
607:            IFIDN   <X>,<RB1>
608:            DB      0D5H
609:            ELSE
610:            IFIDN   <X>,<MB0>
611:            DB      0E5H
612:            ELSE
613:            IFIDN   <X>,<MB1>
614:            DB      0F5H
615:            ELSE
616:            ERROR   <OPERAND ERROR>
617:            EXITM
618:            ENDIF
619:            ENDIF
620:            ENDIF
621:            ENDIF
622:            ENDM
623:    EN      MACRO   X
624:            IFIDN   <X>,<I>
625:            DB      05H
626:            ELSE
627:            IFIDN   <X>,<TCNTI>
628:            DB      25H
629:            ELSE
630:            ERROR   <INVALID OPERAND>
631:            ENDIF
632:            ENDIF
633:            ENDM
634:    DIS     MACRO   X
635:            IFIDN   <X>,<I>
636:            DB      15H
637:            ELSE
638:            IFIDN   <X>,<TCNTI>
639:            DB      35H
640:            ELSE
641:            ERROR   <INVALID OPERAND>
642:            ENDIF
643:            ENDIF
644:            ENDM
645:    ENTO    MACRO   X
646:            IFIDN   <X>,<CLK>
647:            DB      75H
648:            ELSE
649:            ERROR   <INVALID OPERAND>
650:            ENDIF
651:            ENDM
652:    REPEAT  MACRO
653:            @PUT    %@CT
654:            ENDM
655:    UNTIL   MACRO   X,Y,Z
656:            IFE     @CT
657:            ERROR   <REPEAT EXPECTED>
658:            EXITM
659:            ELSE
660:            @CT      DEFL   @CT-1
661:            CHECK1  Z
662:            IF      @FLG
663:            DB      0F0H OR Z         ;;MOV A,Rr
664:            ELSE
665:            DB      23H,Z             ;;MVI A,DATA
666:            ENDIF
667:            DB      37H,17H           ;;CLP A : INX A
668:            CHECK1  X
669:            IF      @FLG
670:            DB      60H OR X          ;;ADD A,Rr
671:            ELSE
672:            DB      03H,X             ;;ADD A,DATA
673:            ENDIF
674:            IFIDN   <EQ>,<Y>
675:            @GET    %@CT
676:            SHORT   96H,@WK           ;;JNZ
677:            ELSE
678:            IFIDN   <NE>,<Y>
679:            @GET    %@CT
680:            SHORT   0C6H,@WK          ;;JZ
681:            ELSE
682:            IFIDN   <GR>,<Y>
683:            @GET    %@CT
684:            SHORT   0E6H,@WK          ;;JNC
685:            ELSE
686:            IFIDN   <GE>,<Y>
687:            @GET    %@CT
688:            @WK0    DEFL    $+4       ;;JZ
689:            SHORT   0C6H,@WK0
690:            SHORT   0E6H,@WK          ;;JNC
691:            ELSE
692:            IFIDN   <LS>,<Y>
693:            @GET    %@CT
694:            SHORT   00F6H,@WK         ;;JC
695:            ELSE
696:            IFIDN   <LE>,<Y>
697:            @GET    %@CT
698:            @WK0    DEFL    $+4       ;;JZ
699:            SHORT   0C6H,@WK0
700:            SHORT   0F6H,@WK          ;;JC
701:            ENDIF
702:            ENDIF
703:            ENDIF
704:            ENDIF
705:            ENDIF
706:            ENDIF
707:            ENDIF
708:            ENDM
709:    PRINTE  MACRO
710:            IF2
711:            IF      @ERR
712:            PRINTE1 %@ERR,< ERROR(s) >
713:            ELSE
714:            PRINTE1 < NO >,< ERROR(S) >
715:            ENDIF
716:            ENDIF
717:            ENDM
718:    PRINTE1 MACRO   X,Y
719:            .PRINTX *X,Y*
720:            ENDM
721:            ;;
722:    @CT     DEFL    0
723:    @ERR    DEFL    0
724:    A       EQU     07H
725:    R0      EQU     08H
726:    R1      EQU     09H
727:    R2      EQU     0AH
728:    R3      EQU     0BH
729:    R4      EQU     0CH
730:    R5      EQU     0DH
731:    R6      EQU     0EH
732:    R7      EQU     0FH
733:    @R0     EQU     00H
734:    @R1     EQU     01H
735:    BUS     EQU     00H
736:    P1      EQU     01H
737:    P2      EQU     02H
738:    P4      EQU     00H
739:    P5      EQU     01H
740:    P6      EQU     02H
741:    P7      EQU     03H
```

トランジスタ技術
SPECIAL No.6

© CQ出版社 1987

1987年11月1日 初版発行
1989年11月1日 第5版発行

発行人 神戸一夫　編集人 蒲生良治
発行所 CQ出版株式会社　〒170 東京都豊島区巣鴨1-14-2
電　話 03(947)6311〜6315
振　替 東京0-10665

(定価は表四に表示してあります)

印刷・製本 三晃印刷株式会社